Package ‘phyloseq’

April 1, 2025

Version 1.50.0

Date 2021-11-29

Title Handling and analysis of high-throughput microbiome census data

Description phyloseq provides a set of classes and tools
to facilitate the import, storage, analysis, and
graphical display of microbiome census data.

Maintainer Paul J. McMurdie <`joey711@gmail.com`>

Author Paul J. McMurdie <`joey711@gmail.com`>,
Susan Holmes <`susan@stat.stanford.edu`>, with
contributions from Gregory Jordan and Scott Chamberlain

License AGPL-3

Imports ade4 (>= 1.7-4), ape (>= 5.0), Biobase (>= 2.36.2), BiocGenerics (>= 0.22.0), biomformat (>= 1.0.0), Biostrings (>= 2.40.0), cluster (>= 2.0.4), data.table (>= 1.10.4), foreach (>= 1.4.3), ggplot2 (>= 2.1.0), igraph (>= 1.0.1), methods (>= 3.3.0), multtest (>= 2.28.0), plyr (>= 1.8.3), reshape2 (>= 1.4.1), scales (>= 0.4.0), vegan (>= 2.5)

Depends R (>= 3.3.0)

Suggests BiocStyle (>= 2.4), DESeq2 (>= 1.16.1), genefilter (>= 1.58), knitr (>= 1.16), magrittr (>= 1.5), metagenomeSeq (>= 1.14), rmarkdown (>= 1.6), testthat (>= 1.0.2)

VignetteBuilder knitr

Enhances doParallel (>= 1.0.10)

biocViews ImmunoOncology, Sequencing, Microbiome, Metagenomics, Clustering, Classification, MultipleComparison, GeneticVariability

URL `http://dx.plos.org/10.1371/journal.pone.0061217`

BugReports `https://github.com/joey711/phyloseq/issues`

Collate 'allClasses.R' 'allPackage.R' 'allData.R' 'as-methods.R' 'show-methods.R' 'plot-methods.R' 'extract-methods.R' 'almostAllAccessors.R' 'otuTable-class.R' 'phyloseq-class.R' 'taxonomyTable-class.R' 'IO-methods.R' 'merge-methods.R' 'multtest-wrapper.R' 'ordination-methods.R' 'transform_filter-methods.R' 'validity-methods.R' 'assignment-methods.R' 'sampleData-class.R' 'extend_vegan.R'

'network-methods.R' 'distance-methods.R'
'deprecated_functions.R' 'extend_DESeq2.R' 'phylo-class.R'
'extend_metagenomeSeq.R'

RoxygenNote 6.1.1

git_url https://git.bioconductor.org/packages/phyloseq

git_branch RELEASE_3_20

git_last_commit 8724974

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-03-31

Contents

`phyloseq-package` *Handling and analysis of high-throughput phylogenetic sequence data.*

Description

There are already several ecology and phylogenetic packages available in R, including the adephylo, vegan, ade4, picante, ape, phangorn, phylobase, and OTUbase packages. These can already take advantage of many of the powerful statistical and graphics tools available in R. However, prior to *phyloseq* a user must devise their own methods for parsing the output of their favorite OTU clustering application, and, as a consequence, there is also no standard within Bioconductor (or R generally) for storing or sharing the suite of related data objects that describe a phylogenetic sequencing project. The phyloseq package seeks to address these issues by providing a related set of S4 classes that internally manage the handling tasks associated with organizing, linking, storing, and analyzing phylogenetic sequencing data. *phyloseq* additionally provides some convenience wrappers for input from common clustering applications, common analysis pipelines, and native implementation of methods that are not available in other R packages.

Author(s)

Paul J. McMurdie II `<mcmurdie@stanford.edu>`

References

`www.stanford.edu/~mcmurdie`

`access` *Universal slot accessor function for phyloseq-class.*

Description

This function is used internally by many accessors and in many functions/methods that need to access a particular type of component data. If something is wrong, or the slot is missing, the expected behavior is that this function will return NULL. Thus, the output can be tested by `is.null` as verification of the presence of a particular data component. Unlike the component-specific accessors (e.g. `otu_table`, or `phy_tree`), the default behavior is not to stop with an error if the desired slot is empty. In all cases this is controlled by the `errorIfNULL` argument, which can be set to `TRUE` if an error is desired.

Usage

```
access(physeq, slot, errorIfNULL=FALSE)
```

Arguments

`physeq`	(Required). `phyloseq-class`.
`slot`	(Required). A character string indicating the slot (not data class) of the component data type that is desired.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `FALSE`.

Value

Returns the component object specified by the argument `slot`. Returns NULL if slot does not exist. Returns `physeq` as-is if it is a component class that already matches the slot name.

See Also

`getslots.phyloseq`, `merge_phyloseq`

Examples

```
#
## data(GlobalPatterns)
## access(GlobalPatterns, "tax_table")
## access(GlobalPatterns, "phy_tree")
## access(otu_table(GlobalPatterns), "otu_table")
## # Should return NULL:
## access(otu_table(GlobalPatterns), "sample_data")
## access(otuTree(GlobalPatterns), "sample_data")
## access(otuSam(GlobalPatterns), "phy_tree")
```

`build_tax_table` *Build a* `tax_table` *from a named possibly-jagged list*

Description

Build a `tax_table` from a named possibly-jagged list

Usage

```
build_tax_table(taxlist)
```

Arguments

`taxlist`	(Required). A list in which each element is a vector of taxonomic assignments named by rank. Every element of every vector must be named by the rank it represents. Every element of the list (every vector) should correspond to a single OTU and be named for that OTU.

Value

A `tax_table` (`taxonomyTable-class`) that has been built from `taxlist`. The OTU names of this output will be the element names of `taxlist`, and a separate taxonomic rank (column) will be included for each unique rank found among the element names of each vector in the list. `NA_character_` is the default value of elements in the `tax_table` for which there is no corresponding information in `taxlist`.

See Also

`import_biom` `import_qiime`

Examples

```
taxvec1 = c("Root", "k__Bacteria", "p__Firmicutes", "c__Bacilli", "o__Bacillales", "f__Staphylococcaceae")
parse_taxonomy_default(taxvec1)
parse_taxonomy_greengenes(taxvec1)
taxvec2 = c("Root;k__Bacteria;p__Firmicutes;c__Bacilli;o__Bacillales;f__Staphylococcaceae")
parse_taxonomy_qiime(taxvec2)
taxlist1 = list(OTU1=parse_taxonomy_greengenes(taxvec1), OTU2=parse_taxonomy_qiime(taxvec2))
taxlist2 = list(OTU1=parse_taxonomy_default(taxvec1), OTU2=parse_taxonomy_qiime(taxvec2))
build_tax_table(taxlist1)
build_tax_table(taxlist2)
```

`capscale.phyloseq` *Constrained Analysis of Principal Coordinates, `capscale`.*

Description

See `capscale` for details. A formula is main input.

Usage

```
capscale.phyloseq(physeq, formula, distance, ...)

## S4 method for signature 'phyloseq,formula,dist'
capscale.phyloseq(physeq, formula,
  distance, ...)

## S4 method for signature 'phyloseq,formula,character'
capscale.phyloseq(physeq, formula,
  distance, ...)
```

Arguments

`physeq`	(Required). Phylogenetic sequencing data (`phyloseq-class`). The data on which you want to perform the ordination.
`formula`	(Required). A `formula`, specifying the input. No need to directly access components. `capscale.phyloseq` understands where to find the abundance table (LHS) and `sample_data` (RHS) from within the phyloseq object.
`distance`	(Required). A `character` string, specifying the name of the dissimilarity (or distance) method supported by the phyloseq `distance` function. Alternatively, a pre-computed `dist`-object can be provided here, in which case it supersedes any use of the `otu_table` in your phyloseq object. Note that `capscale` with Euclidean distances will be identical to `rda` in eigenvalues and in site, species, and biplot scores (except for possible sign reversal). However, it makes no sense to use `capscale` with Euclidean distances, since direct use of `rda` is much more efficient (and supported in the `ordinate` function with `method=="RDA"`) Even with non-Euclidean dissimilarities, the rest of the analysis will be metric and linear.
`...`	(Optional). Additional named arguments passed to `capscale`.

Value

Ordination object defined by `capscale`.

See Also

`plot_ordination`

`rda`

`capscale`

Examples

```
# See other examples at
# http://joey711.github.io/phyloseq/plot_ordination-examples
data(GlobalPatterns)
GP = prune_taxa(names(sort(taxa_sums(GlobalPatterns), TRUE)[1:50]), GlobalPatterns)
ordcap = ordinate(GP, "CAP", "bray", ~SampleType)
plot_ordination(GP, ordcap, "samples", color="SampleType")
```

`cca.phyloseq` *Constrained Correspondence Analysis and Redundancy Analysis.*

Description

This is the internal function that simplifies getting phyloseq data into the constrained ordination functions, `cca` and `rda`. Unlike `capscale.phyloseq`, the formula argument to these methods is optional, and results in an unconstrained ordination.

Usage

```
cca.phyloseq(physeq, formula = NULL, method = "CCA", ...)

## S4 method for signature 'phyloseq,formula'
cca.phyloseq(physeq, formula = NULL,
  method = "CCA", ...)

## S4 method for signature 'otu_table,ANY'
cca.phyloseq(physeq, formula = NULL,
  method = "CCA", ...)

## S4 method for signature 'phyloseq,`NULL`'
cca.phyloseq(physeq, formula = NULL,
  method = "CCA", ...)
```

Arguments

`physeq`	(Required). Phylogenetic sequencing data (`phyloseq-class`). The data on which you want to perform the ordination.
`formula`	(Optional). A `formula`, specifying the contraining variable(s) format, with variable names corresponding to `sample_data` (RHS) from within physeq.
`method`	(Optional). A single `character` string, specifying `"RDA"` or `"CCA"`. Default is `"CCA"`.
`...`	(Optional). Additional named arguments passed to `capscale`.

Value

same output as `cca` or `rda`, respectively.

See Also

`plot_ordination`, `rda`, `cca`

Examples

```
#
# cca.phyloseq(physeq, formula, method, ...)
```

`chunkReOrder` *Chunk re-order a vector so that specified newstart is first.*

Description

Different than relevel.

Usage

```
chunkReOrder(x, newstart = x[[1]])
```

Examples

```
# Typical use-case
# chunkReOrder(1:10, 5)
# # Default is to not modify the vector
# chunkReOrder(1:10)
# # Another example not starting at 1
# chunkReOrder(10:25, 22)
# # Should silently ignore the second element of `newstart`
# chunkReOrder(10:25, c(22, 11))
# # Should be able to handle `newstart` being the first argument already
# # without duplicating the first element at the end of `x`
# chunkReOrder(10:25, 10)
# all(chunkReOrder(10:25, 10) == 10:25)
# # This is also the default
# all(chunkReOrder(10:25) == 10:25)
# # An example with characters
# chunkReOrder(LETTERS, "G")
# chunkReOrder(LETTERS, "B")
# chunkReOrder(LETTERS, "Z")
# # What about when `newstart` is not in `x`? Return x as-is, throw warning.
# chunkReOrder(LETTERS, "g")
```

`data-enterotype` *(Data) Enterotypes of the human gut microbiome (2011)*

Description

Published in Nature in early 2011, this work compared (among other things), the faecal microbial communities from 22 subjects using complete shotgun DNA sequencing. Authors further compared these microbial communities with the faecal communities of subjects from other studies. A total of 280 faecal samples / subjects are represented in this dataset, and 553 genera. The authors claim that the data naturally clumps into three community-level clusters, or “enterotypes”, that are not immediately explained by sequencing technology or demographic features of the subjects, but with potential relevance to understanding human gut microbiota.

Details

abstract from research article (quoted):

Our knowledge of species and functional composition of the human gut microbiome is rapidly increasing, but it is still based on very few cohorts and little is known about variation across the world. By combining 22 newly sequenced faecal metagenomes of individuals from four countries with previously published data sets, here we identify three robust clusters (referred to as enterotypes hereafter) that are not nation or continent specific. We also confirmed the enterotypes in two published, larger cohorts, indicating that intestinal microbiota variation is generally stratified, not continuous. This indicates further the existence of a limited number of well-balanced host-microbial symbiotic states that might respond differently to diet and drug intake. The enterotypes are mostly driven by species composition, but abundant molecular functions are not necessarily provided by abundant species, highlighting the importance of a functional analysis to understand microbial communities. Although individual host properties such as body mass index, age, or gender cannot explain the observed enterotypes, data-driven marker genes or functional modules can be identified for each of these host properties. For example, twelve genes significantly correlate with age and three functional modules with the body mass index, hinting at a diagnostic potential of microbial markers.

(end quote)

Author(s)

Arumugam, M., Raes, J., et al.

References

Arumugam, M., et al. (2011). Enterotypes of the human gut microbiome.

Nature, 473(7346), 174-180.

`http://www.nature.com/doifinder/10.1038/nature09944` See supplemental information for subject data.

OTU-clustered data was downloaded from the publicly-accessible:

`http://www.bork.embl.de/Docu/Arumugam_et_al_2011/downloads.html`

Examples

```
data(enterotype)
ig <- make_network(enterotype, "samples", max.dist=0.3)
plot_network(ig, enterotype, color="SeqTech", shape="Enterotype", line_weight=0.3, label=NULL)
```

`data-esophagus` *(Data) Small example dataset from a human esophageal community (2004)*

Description

Includes just 3 samples, 1 each from 3 subjects. Although the research article mentions 4 subjects, only 3 are included in this dataset.

Details

abstract from research article (quoted):

The esophagus, like other luminal organs of the digestive system, provides a potential environment for bacterial colonization, but little is known about the presence of a bacterial biota or its nature. By using broad-range 16S rDNA PCR, biopsies were examined from the normal esophagus of four human adults. The 900 PCR products cloned represented 833 unique sequences belonging to 41 genera, or 95 species-level operational taxonomic units (SLOTU); 59 SLOTU were homologous with culture-defined bacterial species, 34 with 16S rDNA clones, and two were not homologous with any known bacterial 16S rDNA. Members of six phyla, Firmicutes, Bacteroides, Actinobacteria, Proteobacteria, Fusobacteria, and TM7, were represented. A large majority of clones belong to 13 of the 41 genera (783/900, 87%), or 14 SLOTU (574/900, 64%) that were shared by all four persons. Streptococcus (39%), Prevotella (17%), and Veilonella (14%) were most prevalent. The present study identified 56-79% of SLOTU in this bacterial ecosystem. Most SLOTU of esophageal biota are similar or identical to residents of the upstream oral biota, but the major distinction is that a large majority (82%) of the esophageal bacteria are known and cultivable. These findings provide evidence for a complex but conserved bacterial population in the normal distal esophagus.

(end quote)

A description of the 16S rRNA sequence processing can be found on the mothur-wiki at the link below. A cutoff of 0.10 was used for OTU clustering in that example, and it is taken here as well to create example data, `esophagus`, which was easily imported with the `import_mothur()` function.

Author(s)

Pei et al. `<zhiheng.pei@med.nyu.edu>`

References

Pei, Z., Bini, E. J., Yang, L., Zhou, M., Francois, F., & Blaser, M. J. (2004). Bacterial biota in the human distal esophagus. Proceedings of the National Academy of Sciences of the United States of America, 101(12), 4250-4255. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC384727`

mothur-processed files and the sequence data can be downloaded from a zip-file, along with additional description, from the following URL: `http://www.mothur.org/wiki/Esophageal_community_analysis`

Examples

```
data(esophagus)
UniFrac(esophagus, weighted=TRUE)
# How to re-create the esophagus dataset using import_mothur function
mothlist  <- system.file("extdata", "esophagus.fn.list.gz", package="phyloseq")
```

```
mothgroup <- system.file("extdata", "esophagus.good.groups.gz", package="phyloseq")
mothtree  <- system.file("extdata", "esophagus.tree.gz", package="phyloseq")
show_mothur_cutoffs(mothlist)
cutoff    <- "0.10"
esophman  <- import_mothur(mothlist, mothgroup, mothtree, cutoff)
```

`data-GlobalPatterns` *(Data) Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample (2011)*

Description

Published in PNAS in early 2011. This work compared the microbial communities from 25 environmental samples and three known "mock communities" – a total of 9 sample types – at a depth averaging 3.1 million reads per sample. Authors were able to reproduce diversity patterns seen in many other published studies, while also invesitigating technical issues/bias by applying the same techniques to simulated microbial communities of known composition.

Details

abstract from research article (quoted):

The ongoing revolution in high-throughput sequencing continues to democratize the ability of small groups of investigators to map the microbial component of the biosphere. In particular, the coevolution of new sequencing platforms and new software tools allows data acquisition and analysis on an unprecedented scale. Here we report the next stage in this coevolutionary arms race, using the Illumina GAIIx platform to sequence a diverse array of 25 environmental samples and three known "mock communities" at a depth averaging 3.1 million reads per sample. We demonstrate excellent consistency in taxonomic recovery and recapture diversity patterns that were previously reported on the basis of metaanalysis of many studies from the literature (notably, the saline/nonsaline split in environmental samples and the split between host-associated and free-living communities). We also demonstrate that 2,000 Illumina single-end reads are sufficient to recapture the same relationships among samples that we observe with the full dataset. The results thus open up the possibility of conducting large-scale studies analyzing thousands of samples simultaneously to survey microbial communities at an unprecedented spatial and temporal resolution.

(end quote)

Many thanks to J. Gregory Caporaso for directly providing the OTU-clustered data files for inclusion in this package.

Author(s)

Caporaso, J. G., et al.

References

Caporaso, J. G., et al. (2011). Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample. PNAS, 108, 4516-4522. PMCID: PMC3063599

The primary article can be viewed/downloaded at: `http://www.pnas.org/content/108/suppl.1/4516.short`

See Also

The examples on the phyloseq wiki page for `plot_ordination` show many more examples:

https://github.com/joey711/phyloseq/wiki/plot_ordination

Examples

```
data(GlobalPatterns)
plot_richness(GlobalPatterns, x="SampleType", measures=c("Observed", "Chao1", "Shannon"))
```

`data-soilrep` *(Data) Reproducibility of soil microbiome data (2011)*

Description

Published in early 2011, this work compared 24 separate soil microbial communities under four treatment conditions via multiplexed/barcoded 454-pyrosequencing of PCR-amplified 16S rRNA gene fragments. The authors found differences in the composition and structure of microbial communities between soil treatments. As expected, the soil microbial communities were highly diverse, with a staggering 16,825 different OTUs (species) observed in the included dataset. Interestingly, this study used a larger number of replicates than previous studies of this type, for a total of 56 samples, and the putatively low resampling rate of species between replicated sequencing trials (“OTU overlap”) was a major concern by the authors.

Details

This dataset contains an experiment-level (`phyloseq-class`) object, which in turn contains the taxa-contingency table and soil-treatment table as `otu_table-class` and `sample_data-class` components, respectively.

This data was imported from raw files supplied directly by the authors via personal communication for the purposes of including as an example in the `phyloseq-package`. As this data is sensitive to choices in OTU-clustering parameters, attempts to recreate the `otu_table` from the raw sequencing data may give slightly different results than the table provided here.

abstract from research article (quoted):

To determine the reproducibility and quantitation of the amplicon sequencing-based detection approach for analyzing microbial community structure, a total of 24 microbial communities from a long-term global change experimental site were examined. Genomic DNA obtained from each community was used to amplify 16S rRNA genes with two or three barcode tags as technical replicates in the presence of a small quantity (0.1% wt/wt) of genomic DNA from Shewanella oneidensis MR-1 as the control. The technical reproducibility of the amplicon sequencing-based detection approach is quite low, with an average operational taxonomic unit (OTU) overlap of 17.2%+/-2.3% between two technical replicates, and 8.2%+/-2.3% among three technical replicates, which is most likely due to problems associated with random sampling processes. Such variations in technical replicates could have substantial effects on estimating beta-diversity but less on alpha-diversity. A high variation was also observed in the control across different samples (for example, 66.7-fold for the forward primer), suggesting that the amplicon sequencing-based detection approach could not be quantitative. In addition, various strategies were examined to improve the comparability of amplicon sequencing data, such as increasing biological replicates, and removing singleton sequences and less-representative OTUs across biological replicates. Finally, as expected, various statistical analyses with preprocessed experimental data revealed clear differences in the composition and structure of microbial communities between warming and non-warming, or between clipping and non-clipping. Taken together, these results suggest that amplicon sequencing-based detection is useful in analyzing microbial community structure even though it is not reproducible and quantitative. However, great caution should be taken in experimental design and data interpretation when the amplicon sequencing-based detection approach is used for quantitative analysis of the beta-diversity of microbial communities.

(end quote)

Author(s)

Jizhong Zhou, et al.

References

Zhou, J., Wu, L., Deng, Y., Zhi, X., Jiang, Y.-H., Tu, Q., Xie, J., et al. Reproducibility and quantitation of amplicon sequencing-based detection. The ISME Journal. (2011) 5(8):1303-1313. `doi:10.1038/ismej.2011.11`

The article can be accessed online at `http://www.nature.com/ismej/journal/v5/n8/full/ismej201111a.html`

Examples

```
# Load the data
data(soilrep)
################################################################################
# Alpha diversity (richness) example. Accept null hypothesis:
# No convincing difference in species richness between warmed/unwarmed soils.
################################################################################
# Graphically compare richness between the different treatments.
man.col <- c(WC="red", WU="brown", UC="blue", UU="darkgreen")
plot_richness(soilrep, x="Treatment", color="Treatment", measures=c("Observed", "Chao1", "Shannon"))
```

`decorana` *S3 class placeholder definition (list) for decorana*

Description

The ape package does export a version of its `decorana`-class, partly because it is not really defined formally anywhere. Instead, it is an S3 class extended from the base class, `list` – this is a very common and easy approach – and proper behavior of any method taking an instance of this class requires exact naming conventions for element names of the list components. The phyloseq package does not provide any validity checks that a given phylo instance is valid (conforms to the conventions in the ape package)... yet. If problems arise, this might be considered, and they could be defined judiciously and within phyloseq.

Usage

```
decorana
```

Format

An object of class `decorana` of length 0.

See Also

`decorana`

`dist-class` *An S4 placeholder for the `dist` class.*

Description

See `dist` for details about this type of a distance matrix object.

See Also

`dist`, `setOldClass`

`distance` *Calculate distance, dissimilarity*

Description

Takes a `phyloseq-class` object and method option, and returns a `distance` object suitable for certain ordination methods and other distance-based analyses. Only sample-wise distances are currently supported (the `type` argument), but eventually species-wise (OTU-wise) distances may be supported as well.

Usage

```
distance(physeq, method, type = "samples", ...)

## S4 method for signature 'phyloseq,ANY'
distance(physeq, method)

## S4 method for signature 'otu_table,character'
distance(physeq, method,
  type = "samples", ...)

## S4 method for signature 'phyloseq,character'
distance(physeq, method, type = "samples",
  ...)
```

Arguments

`physeq`	(Required). A `phyloseq-class` or an `otu_table-class` object. The latter is only appropriate for methods that do not require any additional data (one-table). For example, the “wunifrac” option (`UniFrac`) requires `phyloseq-class` that contains both an `otu_table` and a phylogenetic tree (`phylo`).

`method`	(Required). A character string. Provide one of the currently supported options. See `distanceMethodList` for a detailed list of the supported options here, and links to accompanying documentation.
	Note that for the common definition of `Jaccard` distance using the `vegan-package` implementation, an additional argument is needed, with the full call having the form: `distance(physeq, method = "jaccard", binary = TRUE)`
	The following methods are implemented explicitly within the `phyloseq-package`, and accessed by the following `method` options:
	`"unifrac"` Original (unweighted) UniFrac distance, `UniFrac`
	`"wunifrac"` weighted-UniFrac distance, `UniFrac`
	`"dpcoa"` sample-wise distance used in Double Principle Coordinate Analysis, `DPCoA`
	`"jsd"` Jensen-Shannon Divergence, `JSD`
	Alternatively, you can provide a character string that defines a custom distance method, if it has the form described in `designdist`.

`type`	(Optional). A character string. The type of pairwise comparisons being calculated: sample-wise or taxa-wise. The default is `c("samples")`.
`...`	Additional arguments passed on to the appropriate distance function, determined by the `method` argument.

Details

Depending on the `method` argument, `distance()` wraps one of `UniFrac`, `DPCoA`, `JSD`, `vegdist`, `betadiver`, `designdist`, or `dist`.

Value

An object of class “`dist`” suitable for certain ordination methods and other distance-based analyses.

See Also

`plot_ordination`, `UniFrac`, `DPCoA`, `JSD`, `vegdist`, `betadiver`, `designdist`, `dist`.

Examples

```
data(esophagus)
distance(esophagus, "uunifrac") # Unweighted UniFrac
distance(esophagus, "wunifrac") # weighted UniFrac
distance(esophagus, "jaccard", binary = TRUE) # vegdist jaccard
distance(esophagus, "gower") # vegdist option "gower"
distance(esophagus, "g") # designdist method option "g"
distance(esophagus, "minkowski") # invokes a method from the base dist() function.
distance(esophagus, "(A+B-2*J)/(A+B)") # designdist custom distance
distanceMethodList
help("distance")
```

`distanceMethodList` *List of distance method keys supported in* `distance`

Description

Distance methods should be specified by exact string match. Cannot do partial matching for all options, because too many similar options in downstream method dispatch.

Usage

```
distanceMethodList
```

Format

A list of character vectors. Every entry specifies a supported distance method. Names in the list indicate which downstream function is being utilized for further details. Same functions are linked in the itemized list below.

`unifrac` `UniFrac`

`wunifrac` `UniFrac`

`dpcoa` `DPCoA`

`jsd` `JSD`

`manhattan` `vegdist`

`euclidean` `vegdist`

`canberra` `vegdist`

`bray` `vegdist`

`kulczynski` `vegdist`

`jaccard` `vegdist`

`gower` `vegdist`

`altGower` `vegdist`

`morisita` `vegdist`

`horn` `vegdist`

`mountford` `vegdist`

`raup` `vegdist`

`binomial` `vegdist`

`chao` `vegdist`

`cao` `vegdist`

`w` `betadiver`

`-` `betadiver`

`c` `betadiver`

`wb` `betadiver`

`r` `betadiver`

`I` `betadiver`

`e` `betadiver`

`t` `betadiver`

`me` `betadiver`

`j` `betadiver`

`sor` `betadiver`

`m` `betadiver`

`-` `betadiver`

`co` `betadiver`

`cc` `betadiver`

`g` `betadiver`

`-` `betadiver`

`l` `betadiver`

`hk` `betadiver`

`rlb` `betadiver`

`sim` `betadiver`

`gl` `betadiver`

`z` `betadiver`

`maximum` `dist`

`binary` `dist`

`minkowski` `dist`

`ANY` `designdist`

See Also

`distance`

Examples

```
distanceMethodList
```

`DPCoA` *Calculate Double Principle Coordinate Analysis (DPCoA) using phylogenetic distance*

Description

Function uses abundance (`otu_table-class`) and phylogenetic (`phylo`) components of a `phyloseq-class` experiment-level object to perform a Double Principle Coordinate Analysis (DPCoA), relying heavily on the underlying (and more general) function, `dpcoa`. The distance object ultimately provided is the square root of the cophenetic/patristic (`cophenetic.phylo`) distance between the species, which is always Euclidean.

Although this distance is Euclidean, for numerical reasons it will sometimes look non-Euclidean, and a correction will be performed. See `correction` argument.

Usage

```
DPCoA(physeq, correction = cailliez, scannf = FALSE, ...)
```

Arguments

`physeq`	(Required). A `phyloseq-class` object containing, at a minimum, abundance (`otu_table-class`) and phylogenetic (`phylo`) components. As a test, the accessors `otu_table` and `phy_tree` should return an object without error.
`correction`	(Optional). A function. The function must be able to take a non-Euclidean `distance` object, and return a new `distance` object that is Euclidean. If testing a distance object, try `is.euclid`. Although the distance matrix should always be Euclidean, for numerical reasons it will sometimes appear non-Euclidean and a correction method must be applied. Two recommended correction methods are `cailliez` and `lingoes`. The default is `cailliez`, but not for any particularly special reason. If the distance matrix is Euclidian, no correction will be performed, regardless of the value of the `correction` argument.
`scannf`	(Optional). Logical. Default is `FALSE`. This is passed directly to `dpcoa`, and causes a barplot of eigenvalues to be created if `TRUE`. This is not included in ... because the default for `dpcoa` is `TRUE`, although in many expected situations we would want to suppress creating the barplot.
`...`	Additional arguments passed to `dpcoa`.

Value

A dpcoa-class object (see `dpcoa`).

Author(s)

Julia Fukuyama <`julia.fukuyama@gmail.com`>. Adapted for phyloseq by Paul J. McMurdie.

References

Pavoine, S., Dufour, A.B. and Chessel, D. (2004) From dissimilarities among species to dissimilarities among communities: a double principal coordinate analysis. Journal of Theoretical Biology, 228, 523-537.

See Also

`dpcoa`

Examples

```
# # # # # # Esophagus
data(esophagus)
eso.dpcoa <- DPCoA(esophagus)
eso.dpcoa
plot_ordination(esophagus, eso.dpcoa, "samples")
plot_ordination(esophagus, eso.dpcoa, "species")
plot_ordination(esophagus, eso.dpcoa, "biplot")
#
#
# # # # # # GlobalPatterns
```

```
data(GlobalPatterns)
# subset GP to top-150 taxa (to save computation time in example)
keepTaxa <- names(sort(taxa_sums(GlobalPatterns), TRUE)[1:150])
GP       <- prune_taxa(keepTaxa, GlobalPatterns)
# Perform DPCoA
GP.dpcoa <- DPCoA(GP)
plot_ordination(GP, GP.dpcoa, color="SampleType")
```

`envHash2otu_table` *Convert a sequence-sample hash (like ENV file) into an OTU table.*

Description

Parses an ENV-file into a sparse matrix of species-by-sample, where each species-row has only one non-zero value. We call this sparse abundance table the trivial OTU table, where every sequence is treated as a separate species. If a phylogenetic tree is available, it can be submitted with this table as arguments to `tip_glom` to create an object with a non-trivial `otu_table`.

Usage

```
envHash2otu_table(tipSampleTable)
```

Arguments

`tipSampleTable`	(Required). A two-column character table (matrix or data.frame), where each row specifies the sequence name and source sample, consistent with the env-file for the UniFrac server (`http://bmf2.colorado.edu/unifrac/`).

Value

`otu_table`. A trivial OTU table where each sequence is treated as a separate OTU.

References

`http://bmf2.colorado.edu/unifrac/`

See Also

`import_env_file`

`tip_glom`

`otu_table`

Examples

```
#
## fakeSeqNameVec <- paste("seq_", 1:8, sep="")
## fakeSamNameVec <- c(rep("A", 4), rep("B", 4))
## fakeSeqAbunVec <- sample(1:50, 8, TRUE)
## test    <- cbind(fakeSeqNameVec, fakeSamNameVec, fakeSeqAbunVec)
## testotu <- envHash2otu_table( test )
## test    <- cbind(fakeSeqNameVec, fakeSamNameVec)
## testotu <- envHash2otu_table( test )
```

`estimate_richness` *Summarize alpha diversity*

Description

Performs a number of standard alpha diversity estimates, and returns the results as a `data.frame`. Strictly speaking, this function is not only estimating richness, despite its name. It can operate on the cumulative population of all samples in the dataset, or by repeating the richness estimates for each sample individually. NOTE: You must use untrimmed datasets for meaningful results, as these estimates (and even the "observed" richness) are highly dependent on the number of singletons. You can always trim the data later on if needed, just not before using this function.

Usage

```
estimate_richness(physeq, split = TRUE, measures = NULL)
```

Arguments

`physeq`	(Required). `phyloseq-class`, or alternatively, an `otu_table-class`. The data about which you want to estimate the richness.
`split`	(Optional). Logical. Should a separate set of richness estimates be performed for each sample? Or alternatively, pool all samples and estimate richness of the entire set.
`measures`	(Optional). Default is `NULL`, meaning that all available alpha-diversity measures will be included. Alternatively, you can specify one or more measures as a character vector of measure names. Values must be among those supported: `c("Observed", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson", "Fisher")`.

Value

A `data.frame` of the richness estimates, and their standard error.

See Also

Check out the custom plotting function, `plot_richness`, for easily showing the results of different estimates, with method-specific error-bars. Also check out the internal functions borrowed from the vegan package:

`estimateR`

`diversity`

`fisherfit`

Examples

```
## There are many more interesting examples at the phyloseq online tutorials.
## http://joey711.github.com/phyloseq/plot_richness-examples
 data("esophagus")
 # Default is all available measures
 estimate_richness(esophagus)
 # Specify just one:
 estimate_richness(esophagus, measures="Observed")
 # Specify a few:
 estimate_richness(esophagus, measures=c("Observed", "InvSimpson", "Shannon", "Chao1"))
```

`export_env_file` *Export environment (ENV) file for UniFrac Server.*

Description

Creates the environment table that is needed for the original UniFrac algorithm. Useful for cross-checking, or if want to use UniFrac server. Optionally the ENV-formatted table can be returned to the `R` workspace, and the tree component can be exported as Nexus format (Recommended).

Usage

```
export_env_file(physeq, file = "", writeTree = TRUE, return = FALSE)
```

Arguments

`physeq`	(Required). Experiment-level (`phyloseq-class`) object. Ideally this also contains the phylogenetic tree, which is also exported by default.
`file`	(Optional). The file path for export. If not-provided, the expectation is that you will want to set `return` to `TRUE`, and manipulate the ENV table on your own. Default is `""`, skipping the ENV file from being written to a file.
`writeTree`	(Optional). Write the phylogenetic tree as well as the the ENV table. Default is `TRUE`.
`return`	(Optional). Should the ENV table be returned to the R workspace? Default is `FALSE`.

Examples

```
# # Load example data
# data(esophagus)
# export_env_file(esophagus, "~/Desktop/esophagus.txt")
```

`export_mothur_dist` *Export a distance object as* `.names` *and* `.dist` *files for mothur*

Description

The purpose of this function is to allow a user to easily export a distance object as a pair of files that can be immediately imported by mothur for OTU clustering and related analysis. A distance object can be created in `R` in a number of ways, including via cataloguing the cophentic distances of a tree object.

Usage

```
export_mothur_dist(x, out=NULL, makeTrivialNamesFile=NULL)
```

Arguments

`x`	(Required). A `"dist"` object, or a symmetric matrix.
`out`	(Optional). The desired output filename for the `.dist` file, OR left `NULL`, the default, in which case the mothur-formated distance table is returned to `R` standard out.
`makeTrivialNamesFile`	(Optional). Default `NULL`. The desired name of the `.names` file. If left `NULL`, the file name will be a modified version of the `out` argument.

Value

A character vector of the different cutoff values contained in the file. For a given set of arguments to the `cluster()` command from within *mothur*, a number of OTU-clustering results are returned in the same list file. The exact cutoff values used by *mothur* can vary depending on the input data. This simple function returns the cutoffs that were actually included in the *mothur* output. This an important extra step prior to importing the OTUs with the `import_mothur_otulist()` function.

Examples

```
#
data(esophagus)
myDistObject <- as.dist(ape::cophenetic.phylo(phy_tree(esophagus)))
export_mothur_dist(myDistObject)
```

`filterfun_sample` *A sample-wise filter function builder analogous to* `filterfun`.

Description

See the `filterfun`, from the Bioconductor repository, for a taxa-/gene-wise filter (and further examples).

Usage

```
filterfun_sample(...)
```

Arguments

`...`	A comma-separated list of functions.

Value

An enclosure (function) that itself will return a logical vector, according to the functions provided in the argument list, evaluated in order. The output of filterfun_sample is appropriate for the 'flist' argument to the genefilter_sample method.

See Also

`filterfun`, `genefilter_sample`

Examples

```
# Use simulated abundance matrix
set.seed(711)
testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
f1  <- filterfun_sample(topk(2))
wh1 <- genefilter_sample(testOTU, f1, A=2)
wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
prune_taxa(wh1, testOTU)
prune_taxa(wh2, testOTU)
```

`filter_taxa` *Filter taxa based on across-sample OTU abundance criteria*

Description

This function is directly analogous to the `genefilter` function for microarray filtering, but is used for filtering OTUs from phyloseq objects. It applies an arbitrary set of functions — as a function list, for instance, created by `filterfun` — as across-sample criteria, one OTU at a time. It takes as input a phyloseq object, and returns a logical vector indicating whether or not each OTU passed the criteria. Alternatively, if the `"prune"` option is set to `FALSE`, it returns the already-trimmed version of the phyloseq object.

Usage

```
filter_taxa(physeq, flist, prune=FALSE)
```

Arguments

`physeq`	(Required). A `phyloseq-class` object that you want to trim/filter.
`flist`	(Required). A function or list of functions that take a vector of abundance values and return a logical. Some canned useful function types are included in the `genefilter`-package.
`prune`	(Optional). A logical. Default `FALSE`. If `TRUE`, then the function returns the pruned `phyloseq-class` object, rather than the logical vector of taxa that passed the filter.

Value

A logical vector equal to the number of taxa in physeq. This can be provided directly to `prune_taxa` as first argument. Alternatively, if `prune==TRUE`, the pruned `phyloseq-class` object is returned instead.

See Also

`filterfun`, `genefilter_sample`, `filterfun_sample`

Examples

```
data("enterotype")
require("genefilter")
flist    <- filterfun(kOverA(5, 2e-05))
ent.logi <- filter_taxa(enterotype, flist)
ent.trim <- filter_taxa(enterotype, flist, TRUE)
identical(ent.trim, prune_taxa(ent.logi, enterotype))
identical(sum(ent.logi), ntaxa(ent.trim))
filter_taxa(enterotype, flist, TRUE)
```

`fix_phylo` *Method for fixing problems with phylo-class trees in phyloseq*

Description

For now this only entails replacing each missing (`NA`) branch-length value with 0.0.

Usage

```
fix_phylo(tree)

## S4 method for signature 'phylo'
fix_phylo(tree)
```

`gapstat_ord` *Estimate the gap statistic on an ordination result*

Description

This is a wrapper for the `clusGap` function, expecting an ordination result as the main data argument.

Usage

```
gapstat_ord(ord, axes = c(1:2), type = "sites",
  FUNcluster = function(x, k) {     list(cluster = pam(x, k, cluster.only
  = TRUE)) }, K.max = 8, ...)
```

Arguments

`ord`	(Required). An ordination object. The precise class can vary. Any ordination classes supported internally by the phyloseq package should work, ultimately by passing to the `scores` function or its internal extensions in phyloseq.
`axes`	(Optional). The ordination axes that you want to include.
`type`	(Optional). One of `"sites"` (the vegan package label for samples) or `"species"` (the vegan package label for OTUs/taxa). Default is `"sites"`.

`FUNcluster`	(Optional). This is passed to `clusGap`. The documentation is copied here for convenience: a function which accepts as first argument a (data) matrix like `x`, second argument, say (the number of desired clusters) `k`, where `k` >= 2, and returns a list with a component named (or shortened to) cluster which is a vector of length `n = nrow(x)` of integers in `1:k` determining the clustering or grouping of the `n` observations. The default value is the following function, which wraps partitioning around medoids, `pam`: `function(x, k){list(cluster = pam(x, k, cluster.only=TRUE))}` Any function that has these input/output properties (performing a clustering) will suffice. The more appropriate the clustering method, the better chance your gap statistic results will be useful.
`K.max`	(Optional). A single positive integer value. It indicates the maximum number of clusters that will be considered. Value must be at least two. This is passed to `clusGap`.
`...`	(Optional). Additional named parameters passed on to `clusGap`. For example, the `method` argument provides for extensive options regarding the method by which the "optimal" number of clusters is computed from the gap statistics (and their standard deviations). See the `clusGap` documentation for more details.

Value

An object of S3 class `"clusGap"`, basically a list with components. See the `clusGap` documentation for more details.

Examples

```
data("soilrep")
sord  = ordinate(soilrep, "PCoA", "bray")
# Evaluate axes with scree plot
plot_scree(sord)
# Gap Statistic
gs = gapstat_ord(sord, axes=1:3, verbose=FALSE)
# plot_ordination(soilrep, sord,  color="Treatment")
plot_clusgap(gs)
print(gs, method="Tibs2001SEmax")
```

`genefilter_sample` *Filter OTUs with arbitrary function, sample-wise.*

Description

A general OTU trimming function for selecting OTUs that satisfy some criteria within the distribution of each sample, and then also an additional criteria for number of samples that must pass. This is a genefilter-like function that only considers sample-wise criteria. The number of acceptable samples is used as the final criteria (set by the argument `A`) to determine whether or not the taxa should be retained (`TRUE`) or not (`FALSE`). Just like with genefilter, a logical having length equal to nrow()/`ntaxa` is returned, indicating which should be kept. This output can be provided directly to OTU trimming function, `prune_taxa`. By contrast, `genefilter`, of the genefilter package in Bioconductor, works only on the rows of a matrix. Note that, because `otu_table-class` inherits directly from the `matrix-class`, an unmodified otu_table can be provided to `genefilter`, but be mindful of the orientation of the otu_table (use `taxa_are_rows`), and transpose (`t`) if needed.

Usage

```
genefilter_sample(X, flist, A=1)

## S4 method for signature 'matrix'
genefilter_sample(X, flist, A = 1)

## S4 method for signature 'otu_table'
genefilter_sample(X, flist, A = 1)

## S4 method for signature 'phyloseq'
genefilter_sample(X, flist, A = 1)
```

Arguments

`X`	The object that needs trimming. Can be matrix, otu_table, or higher- order phyloseq classes that contain an otu_table.
`flist`	An enclosure object, typically created with `filterfun_sample`
`A`	An integer. The number of samples in which a taxa / OTUs passed the filter for it to be labeled TRUE in the output logical vector.

Value

A logical vector with names equal to taxa_names (or rownames, if matrix).

See Also

`genefilter`, `filterfun_sample`, `t`, `prune_taxa`

Examples

```
#
## testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
## f1  <- filterfun_sample(topk(2))
## wh1 <- genefilter_sample(testOTU, f1, A=2)
## wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
## prune_taxa(wh1, testOTU)
## prune_taxa(wh2, testOTU)
##
## tax_table1 <- tax_table(matrix("abc", 5, 5))
## prune_taxa(wh1, tax_table1)
## prune_taxa(wh2, tax_table1)
```

`get.component.classes` *Show the component objects classes and slot names.*

Description

There are no arguments to this function. It returns a named character when called, which can then be used for tests of component data types, etc.

Usage

```
get.component.classes()
```

Value

a character vector of the component objects classes, where each element is named by the corresponding slot name in the phyloseq-class.

Examples

```
#
#get.component.classes()
```

`getslots.phyloseq` *Return the non-empty slot names of a phyloseq object.*

Description

Like `getSlots`, but returns the class name if argument is component data object.

Usage

```
getslots.phyloseq(physeq)
```

Arguments

`physeq`	A `phyloseq-class` object. If `physeq` is a component data class, then just returns the class of `physeq`.

Value

identical to getSlots. A named character vector of the slot classes of a particular S4 class, where each element is named by the slot name it represents. If `physeq` is a component data object, then a vector of length (1) is returned, named according to its slot name in the `phyloseq-class`.

See Also

merge_phyloseq

Examples

```
#
 data(GlobalPatterns)
 getslots.phyloseq(GlobalPatterns)
 data(esophagus)
 getslots.phyloseq(esophagus)
```

`get_sample` *Returns all abundance values for species* `i`.

Description

This is a simple accessor function for investigating a single species-of-interest.

Usage

```
get_sample(physeq, i)

## S4 method for signature 'otu_table'
get_sample(physeq, i)

## S4 method for signature 'phyloseq'
get_sample(physeq, i)
```

Arguments

physeq	(Required). `otu_table-class`, or `phyloseq-class`.
i	(Required). A single taxa/species/OTU ID for which you want to know the abundance in each sample.

Value

An integer vector of the abundance values for each sample in `physeq` for species `i`

See Also

`get_taxa` `taxa_names` `sample_names`

Examples

```
data(esophagus)
taxa_names(esophagus)
get_sample(esophagus, "59_5_19")
```

`get_taxa` *Returns all abundance values of sample* `i`.

Description

This is a simple accessor function for investigating a single sample-of-interest.

Usage

```
get_taxa(physeq, i)

## S4 method for signature 'otu_table'
get_taxa(physeq, i)

## S4 method for signature 'phyloseq'
get_taxa(physeq, i)
```

Arguments

`physeq`	(Required). `otu_table-class`, or `phyloseq-class`.
`i`	(Required). A single sample for which you want to know the abundance of each species. Can be integer for index value, or sample name.

Value

An integer vector of the abundance values for each species in physeq for sample `i`

See Also

`get_sample` `taxa_names` `sample_names`

Examples

```
data(esophagus)
sample_names(esophagus)
get_taxa(esophagus, "B")
```

`get_taxa_unique` *Get a unique vector of the observed taxa at a particular taxonomic rank*

Description

This is a simple accessor function to make it more convenient to determine the different taxa present for a particular taxonomic rank in a given `phyloseq-class` object.

Usage

```
get_taxa_unique(physeq, taxonomic.rank=rank_names(physeq)[1], errorIfNULL=TRUE)
```

Arguments

`physeq`	(Required). `taxonomyTable-class`, or `phyloseq-class`.
`taxonomic.rank`	(Optional). Character. The taxonomic rank to use. Must select from the set indicated by `get_taxa_unique`. Default is to take the first column of the `taxonomyTable` component.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

Character vector. Unique vector of the observed taxa at a particular taxonomic rank

See Also

`get_taxa` `taxa_names` `sample_names`

Examples

```
data(enterotype)
get_taxa_unique(enterotype)
data(GlobalPatterns)
get_taxa_unique(GlobalPatterns, "Family")
```

`get_variable` *Get the values for a particular variable in sample_data*

Description

This is a simple accessor function for streamlining access to values/vectors/factors/etc contained in the sample_data.

Usage

```
get_variable(physeq, varName)
```

Arguments

`physeq`	(Required). `sample_data-class`, or `phyloseq-class`.
`varName`	(Required). Character string of the variable name in `sample_data`. Use `sample_variables(physeq` for available variables in your object.

Value

Data. The clas of the data depends on what the contents of sample_data.

See Also

`get_taxa` `taxa_names` `sample_names`

`sample_variables`

Examples

```
# Load the GlobalPatterns dataset into the workspace environment
data(GlobalPatterns)
# Look at the different values for SampleType
get_variable(GlobalPatterns, "SampleType")
```

`import` *Universal import method (wrapper) for phyloseq-package*

Description

A user must still understand the additional arguments required for each type of import data. Those arguments are described in detail at the tool-specific `import_*` links below. Each clustering tool / package / pipeline has its own idiosyncratic set of file names / types, and it remains the responsibility of the user to understand which file-path should be provided to each argument for the particular importing submethod. This method merely provides a central documentation and method-name, and the arguments are passed along as-is.

Usage

```
import(pipelineName, ...)
```

Arguments

`pipelineName`	(Required). Character string. The name of the analysis tool / pipeline / package that created the OTU-cluster data or other data that you now want to import. Current options are `c("mothur", "pyrotagger", "QIIME", "RDP")`, and only the first letter is necessary.
`...`	(Required). Additional named arguments providing file paths, and possible other paramaters to the desired tool-specific import function.

Value

In most cases a `phyloseq-class` will be returned, though the included component data will vary by pipeline/tool, and also by the types of data files provided. The expected behavior is to return the most-comprehensive object possible, given the provided arguments and pipeline/tool.

References

BIOM: `http://www.biom-format.org/`

mothur: `http://www.mothur.org/wiki/Main_Page`

PyroTagger: `http://pyrotagger.jgi-psf.org/`

QIIME: `http://qiime.org/`

RDP pipeline: `http://pyro.cme.msu.edu/index.jsp`

See Also

For BIOM format, see: `import_biom`

For mothur, see: `import_mothur`

Separate tools for mothur are also: `show_mothur_cutoffs` `import_mothur_dist` `export_mothur_dist`

For PyroTagger, see: `import_pyrotagger_tab`

For QIIME legacy format, see: `import_qiime`

For RDP pipeline, see: `import_RDP_cluster`

`import_RDP_otu`

Examples

```
## See documentation of a specific import function
```

`import_biom` *Import phyloseq data from biom-format file*

Description

New versions of QIIME produce a more-comprehensive and formally-defined JSON file format, called biom file format:

Usage

```
import_biom(BIOMfilename,
 treefilename=NULL, refseqfilename=NULL, refseqFunction=readDNAStringSet, refseqArgs=NULL,
  parseFunction=parse_taxonomy_default, parallel=FALSE, version=1.0, ...)
```

Arguments

`BIOMfilename`	(Required). A character string indicating the file location of the BIOM formatted file. This is a JSON formatted file, specific to biological datasets, as described in `http://www.qiime.org/svn_documentation/documentation/biom_format.html`the biom-format home page. In principle, this file should include you OTU abundance data (OTU table), your taxonomic classification data (taxonomy table), as well as your sample data, for instance what might be in your "sample map" in QIIME. A phylogenetic tree is not yet supported by biom-format, and so is a separate argument here. If, for some reason, your biom-format file is missing one of these mentioned data types but you have it in a separate file, you can first import the data that is in the biom file using this function, `import_biom`, and then "merge" the remaining data after you have imported with other tools using the relatively general-purpose data merging function called `merge_phyloseq`.
`treefilename`	(Optional). Default value is `NULL`. A file representing a phylogenetic tree or a `phylo` object. Files can be NEXUS or Newick format. See `read_tree` for more details. Also, if using a recent release of the GreenGenes database tree, try the `read_tree_greengenes` function – this should solve some issues specific to importing that tree. If provided, the tree should have the same OTUs/tip-labels as the OTUs in the other files. Any taxa or samples missing in one of the files is removed from all. As an example from the QIIME pipeline, this tree would be a tree of the representative 16S rRNA sequences from each OTU cluster, with the number of leaves/tips equal to the number of taxa/species/OTUs, or the complete reference database tree that contains the OTU identifiers of every OTU in your abundance table. Note that this argument can be a tree object (`phylo`-class) for cases where the tree has been — or needs to be — imported separately, as in the case of the GreenGenes tree mentioned earlier (coderead_tree_greengenes).
`refseqfilename`	(Optional). Default `NULL`. The file path of the biological sequence file that contains at a minimum a sequence for each OTU in the dataset. Alternatively, you may provide an already-imported `XStringSet` object that satisfies this condition. In either case, the `names` of each OTU need to match exactly the `taxa_names` of the other components of your data. If this is not the case, for example if the

	data file is a FASTA format but contains additional information after the OTU name in each sequence header, then some additional parsing is necessary, which you can either perform separately before calling this function, or describe explicitly in a custom function provided in the (next) argument, `refseqFunction`. Note that the `XStringSet` class can represent any arbitrary sequence, including user-defined subclasses, but is most-often used to represent RNA, DNA, or amino acid sequences. The only constraint is that this special list of sequences has exactly one named element for each OTU in the dataset.
`refseqFunction`	(Optional). Default is `readDNAStringSet`, which expects to read a fasta-formatted DNA sequence file. If your reference sequences for each OTU are amino acid, RNA, or something else, then you will need to specify a different function here. This is the function used to read the file connection provided as the the previous argument, `refseqfilename`. This argument is ignored if `refseqfilename` is already a `XStringSet` class.
`refseqArgs`	(Optional). Default `NULL`. Additional arguments to `refseqFunction`. See `XStringSet-io` for details about additional arguments to the standard read functions in the Biostrings package.
`parseFunction`	(Optional). A function. It must be a function that takes as its first argument a character vector of taxonomic rank labels for a single OTU and parses and names each element (an optionally removes unwanted elements). Further details and examples of acceptable functions are provided in the documentation for `parse_taxonomy_default`. There are many variations on taxonomic nomenclature, and naming conventions used to store that information in various taxonomic databases and phylogenetic assignment algorithms. A popular database, `http://greengenes.lbl.gov/cgi-bin/nph-index.cgi`greengenes, has its own custom parsing function provided in the phyloseq package, `parse_taxonomy_greengenes`, and more can be contributed or posted as code snippets as needed. They can be custom-defined by a user immediately prior to the the call to `import_biom`, and this is a suggested first step to take when trouble-shooting taxonomy-related errors during file import.
`parallel`	(Optional). Logical. Wrapper option for `.parallel` parameter in `plyr-package` functions. If `TRUE`, apply parsing functions in parallel, using parallel backend provided by `foreach` and its supporting backend packages. One caveat, plyr-parallelization currently works most-cleanly with `multicore`-like backends (Mac OS X, Unix?), and may throw warnings for SNOW-like backends. See the example below for code invoking multicore-style backend within the `doParallel` package. Finally, for many datasets a parallel import should not be necessary because a serial import will be just as fast and the import is often only performed one time; after which the data should be saved as an RData file using the `save` function.
`version`	(Optional). Numeric. The expected version number of the file. As the BIOM format evolves, version-specific importers may be available by adjusting the version value. Default is `1.0`. Not yet implemented. Parsing of the biom-format is done mostly by the biom package now available in CRAN.
`...`	Additional parameters passed on to `read_tree`.

Details

“The biom file format (canonically pronounced ‘biome’) is designed to be a general-use format for representing counts of observations in one or more biological samples. BIOM is a recognized standard for the Earth Microbiome Project and is a Genomics Standards Consortium candidate project."

`http://biom-format.org/`

Value

A `phyloseq-class` object.

References

biom-format

See Also

`import`

`import_qiime`

`read_tree`

`read_tree_greengenes`

`read_biom`

`biom_data`

`sample_metadata`

`observation_metadata`

`XStringSet-io`

Examples

```
# An included example of a rich dense biom file
rich_dense_biom  <- system.file("extdata", "rich_dense_otu_table.biom",  package="phyloseq")
import_biom(rich_dense_biom,  parseFunction=parse_taxonomy_greengenes)
# An included example of a sparse dense biom file
rich_sparse_biom <- system.file("extdata", "rich_sparse_otu_table.biom", package="phyloseq")
import_biom(rich_sparse_biom, parseFunction=parse_taxonomy_greengenes)
# # # Example code for importing large file with parallel backend
# library("doParallel")
# registerDoParallel(cores=6)
# import_biom("my/file/path/file.biom", parseFunction=parse_taxonomy_greengenes, parallel=TRUE)
```

`import_env_file` *Read a UniFrac-formatted ENV file.*

Description

Convenience wrapper function to read the environment-file, as formatted for input to the UniFrac server (`http://bmf2.colorado.edu/unifrac/`). The official format of these files is that each row specifies (in order) the sequence name, source sample, and (optionally) the number of times the sequence was observed.

Usage

```
import_env_file(envfilename, tree=NULL, sep="\t", ...)
```

Arguments

`envfilename`	(Required). A charater string of the ENV filename (relative or absolute)
`tree`	(Optional). `phylo-class` object to be paired with the output otu_table.
`sep`	A character string indicating the delimiter used in the file. The default is `"\t"`.
`...`	Additional parameters passed on to `read.table`.

Value

An `otu_table-class`, or `phyloseq-class` if a `phylo-class` argument is provided to `tree`.

References

`http://bmf2.colorado.edu/unifrac/`

See Also

`import`

`tip_glom`

Examples

```
# import_env_file(myEnvFile, myTree)
```

`import_mothur` *General function for importing mothur data files into phyloseq.*

Description

Technically all parameters are optional, but if you don't provide any file connections, then nothing will be returned. While the `list` and `group` files are the first two arguments for legacy-compatibility reasons, we don't recommend that you use these file types with modern (large) datasets. They are comically inefficient, as they store the name of every sequencing read in both files. The *mothur* package provides conversions utilities to create other more-efficient formats, which we recommend, like the shared file for an OTU table. Alternatively, mothur also provides a utility to create a biom-format file that is independent of OTU clustering platform. Biom-format files should be imported not with this function, but with `import_biom`. The resulting objects after import should be `identical` in R.

Usage

```
import_mothur(mothur_list_file = NULL, mothur_group_file = NULL,
  mothur_tree_file = NULL, cutoff = NULL, mothur_shared_file = NULL,
  mothur_constaxonomy_file = NULL,
  parseFunction = parse_taxonomy_default)
```

Arguments

`mothur_list_file`	(Optional). The list file name / location produced by *mothur*.
`mothur_group_file`	(Optional). The name/location of the group file produced by *mothur*'s `make.group()` function. It contains information about the sample source of individual sequences, necessary for creating a species/taxa abundance table (`otu_table`). See `http://www.mothur.org/wiki/Make.group`
`mothur_tree_file`	(Optional). A tree file, presumably produced by *mothur*, and readable by `read_tree`. The file probably has extension `".tree"`.
`cutoff`	(Optional). A character string indicating the cutoff value, (or `"unique"`), that matches one of the cutoff-values used to produce the OTU clustering results contained within the list-file created by *mothur* (and specified by the `mothur_list_file` argument). The default is to take the largest value among the cutoff values contained in the list file. If only one cutoff is included in the file, it is taken and this argument does not need to be specified. Note that the `cluster()` function within the *mothur* package will often produce a list file with multiple cutoff values, even if a specific cutoff is specified. It is suggested that you check which cutoff values are available in a given list file using the `show_mothur_cutoffs` function.
`mothur_shared_file`	(Optional). A shared file produced by *mothur*.
`mothur_constaxonomy_file`	(Optional). A consensus taxonomy file produced by *mothur*.
`parseFunction`	(Optional). A specific function used for parsing the taxonomy string. See `parse_taxonomy_default` for an example. If the default is used, this function expects a semi-colon delimited taxonomy string, with no additional rank specifier. A common taxonomic database is GreenGenes, and in recent versions its taxonomy entries include a prefix, which is best cleaved and used to precisely label the ranks (`parse_taxonomy_greengenes`).

Value

The object class depends on the provided arguments. A phyloseq object is returned if enough data types are provided. If only one data component can be created from the data, it is returned.

FASTER (recommended for larger data sizes):

If only a `mothur_constaxonomy_file` is provided, then a `taxonomyTable-class` object is returned.

If only a `mothur_shared_file` is provided, then an `otu_table` object is returned.

SLOWER (but fine for small file sizes):

The list and group file formats are extremely inefficient for large datasets, and they are not recommended. The mothur software provides tools for converting to other file formats, such as a so-called “shared” file. You should provide a shared file, or group/list files, but not both at the same time. If only a list and group file are provided, then an `otu_table` object is returned. Similarly, if only a list and tree file are provided, then only a tree is returned (`phylo`-class).

References

http://www.mothur.org/wiki/Main_Page

Schloss, P.D., et al., Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. Appl Environ Microbiol, 2009. 75(23):7537-41.

Examples

```
# # The following example assumes you have downloaded the esophagus example
# # dataset from the mothur wiki:
# # "http://www.mothur.org/wiki/Esophageal_community_analysis"
# # "http://www.mothur.org/w/images/5/55/Esophagus.zip"
# # The path on your machine may (probably will) vary
# mothur_list_file  <- "~/Downloads/mothur/Esophagus/esophagus.an.list"
# mothur_group_file <- "~/Downloads/mothur/Esophagus/esophagus.good.groups"
# mothur_tree_file  <- "~/Downloads/mothur/Esophagus/esophagus.tree"
# # # Actual examples follow:
# show_mothur_cutoffs(mothur_list_file)
# test1 <- import_mothur(mothur_list_file, mothur_group_file, mothur_tree_file)
# test2 <- import_mothur(mothur_list_file, mothur_group_file, mothur_tree_file, cutoff="0.02")
# # Returns just a tree
# import_mothur(mothur_list_file, mothur_tree_file=mothur_tree_file)
# # Returns just an otu_table
# import_mothur(mothur_list_file, mothur_group_file=mothur_group_file)
# # Returns an error
# import_mothur(mothur_list_file)
# # Should return an "OMG, you must provide the list file" error
# import_mothur()
```

`import_mothur_constaxonomy` *Import mothur constaxonomy file and return a taxonomyTable*

Description

Import mothur constaxonomy file and return a taxonomyTable

Usage

```
import_mothur_constaxonomy(mothur_constaxonomy_file,
  parseFunction = parse_taxonomy_default)
```

Arguments

`mothur_constaxonomy_file`	(Required). A consensus taxonomy file produced by *mothur*.
`parseFunction`	(Optional). A specific function used for parsing the taxonomy string. See `parse_taxonomy_default` for an example. If the default is used, this function expects a semi-colon delimited taxonomy string, with no additional rank specifier. A common taxonomic database is GreenGenes, and for recent versions its taxonomy includes a prefix, which is best cleaved and used to precisely label the ranks (`parse_taxonomy_greengenes`).

Value

An `taxonomyTable-class` object.

See Also

`import_mothur`

`tax_table`

`phyloseq`

`import_mothur_dist` *Import mothur-formatted distance file*

Description

The mothur application will produce a file containing the pairwise distances between all sequences in a dataset. This distance matrix can be the basis for OTU cluster designations. R also has many built-in or off-the-shelf tools for dealing with distance matrices.

Usage

```
import_mothur_dist(mothur_dist_file)
```

Arguments

`mothur_dist_file`	Required. The distance file name / location produced by *mothur*.

Value

A distance matrix object describing all sequences in a dataset.

See Also

`import_mothur`

Examples

```
# # Take a look at the dataset shown here as an example:
# # "http://www.mothur.org/wiki/Esophageal_community_analysis"
# # find the file ending with extension ".dist", download to your system
# # The location of your file may vary
# mothur_dist_file <- "~/Downloads/mothur/Esophagus/esophagus.dist"
# myNewDistObject  <- import_mothur_dist(mothur_dist_file)
```

`import_mothur_groups` *Parse mothur group file into a simple hash table.*

Description

The data.frame object returned by this function is not immediately useable by other *phyloseq* functions, and must be first parsed in conjunction with a separate *mothur* `"list"` file. This function is made accessible to *phyloseq* users for troubleshooting and inspection, but the `link{import_mothur()}` function is suggested if the goal is to import the OTU clustering results from *mothur* into *phyloseq*. You will need both a group file and a list file for that end.

Usage

```
import_mothur_groups(mothur_group_file)
```

Arguments

`mothur_group_file`	A character string indicating the location of the *mothur*-produced group file in which the sample-source of each sequence is recorded. See `http://www.mothur.org/wiki/Make.gr`

Value

A data.frame that is effectively a hash table between sequence names and their sample source.

See Also

`import_mothur`

`import_mothur_otulist` *Import mothur list file and return as list object in R.*

Description

This is a user-available module of a more comprehensive function for importing OTU clustering/abundance data using the *mothur* package. The list object returned by this function is not immediately useable by other *phyloseq* functions, and must be first parsed in conjunction with a separate *mothur* `"group"` file. This function is made accessible to *phyloseq* users for troubleshooting and inspection, but the `link{import_mothur()}` function is suggested if the goal is to import the OTU clustering results from *mothur* into *phyloseq*.

Usage

```
import_mothur_otulist(mothur_list_file, cutoff=NULL)
```

Arguments

`mothur_list_file`	The list file name and/or location as produced by *mothur*.
`cutoff`	A character string indicating the cutoff value, (or `"unique"`), that matches one of the cutoff-values used to produce the OTU clustering results contained within the list-file created by *mothur*. The default is to take the largest value among the cutoff values contained in the list file. If only one cutoff is included in the file, it is taken and this argument does not need to be specified. Note that the `cluster()` function within the *mothur* package will often produce a list file with multiple cutoff values, even if a specific cutoff is specified. It is suggested that you check which cutoff values are available in a given list file using the `show_mothur_cutoffs` function.

Value

A list, where each element is a character vector of 1 or more sequence identifiers, indicating how each sequence from the original data is clustered into OTUs by *mothur*. Note that in some cases this is highly dependent on the choice for `cutoff`.

See Also

`show_mothur_cutoffs`, `import_mothur`

`import_mothur_otu_table`

Import mothur list and group files and return an otu_table

Description

Import mothur list and group files and return an otu_table

Usage

```
import_mothur_otu_table(mothur_list_file, mothur_group_file, cutoff=NULL)
```

Arguments

`mothur_list_file`	The list file name and/or location as produced by *mothur*.
`mothur_group_file`	The name/location of the group file produced by *mothur*'s `make.group()` function. It contains information about the sample source of individual sequences, necessary for creating a species/taxa abundance table (`otu_table`). See `http://www.mothur.org/wi`
`cutoff`	A character string indicating the cutoff value, (or `"unique"`), that matches one of the cutoff-values used to produce the OTU clustering results contained within the list-file created by *mothur* (and specified by the `mothur_list_file` argument). The default is to take the largest value among the cutoff values contained in the list file. If only one cutoff is included in the file, it is taken and this argument does not need to be specified. Note that the `cluster()` function within the *mothur* package will often produce a list file with multiple cutoff values, even if a specific cutoff is specified. It is suggested that you check which cutoff values are available in a given list file using the `show_mothur_cutoffs` function.

Value

An `otu_table` object.

See Also

`import_mothur`

`import_mothur_shared` *Import mothur shared file and return an otu_table*

Description

Import mothur shared file and return an otu_table

Usage

```
import_mothur_shared(mothur_shared_file, cutoff = NULL)
```

Arguments

`mothur_shared_file`	(Required). A shared file produced by *mothur*.

Value

An `otu_table` object.

See Also

`import_mothur`

`import_pyrotagger_tab` *Imports a tab-delimited version of the pyrotagger output file.*

Description

PyroTagger is a web-server that takes raw, barcoded 16S rRNA amplicon sequences and returns an excel spreadsheet (`".xls"`) with both abundance and taxonomy data. It also includes some confidence information related to the taxonomic assignment.

Usage

```
import_pyrotagger_tab(pyrotagger_tab_file,
strict_taxonomy=FALSE, keep_potential_chimeras=FALSE)
```

Arguments

`pyrotagger_tab_file`
: (Required). A character string. The name of the tab-delimited pyrotagger output table.

`strict_taxonomy`
: (Optional). Logical. Default `FALSE`. Should the taxonomyTable component be limited to just taxonomic data? Default includes all fields from the pyrotagger file.

`keep_potential_chimeras`
: (Optional). Logical. Default `FALSE`. The pyrotagger output also includes OTUs that are tagged by pyrotagger as likely chimeras. These putative chimeric OTUs can be retained if set to `TRUE`. The putative chimeras are excluded by default.

Details

PyroTagger is created and maintained by the Joint Genome Institute at `"http://pyrotagger.jgi-psf.org/"`

The typical output form PyroTagger is a spreadsheet format `".xls"`, which poses additional import challenges. However, virtually all spreadsheet applications support the `".xls"` format, and can further export this file in a tab-delimited format. It is recommended that you convert the xls-file without any modification (as tempting as it might be once you have loaded it) into a tab-delimited text file. Deselect any options to encapsulate fields in quotes, as extra quotes around each cell's contents might cause problems during file processing. These quotes will also inflate the file-size, so leave them out as much as possible, while also resisting any temptation to modify the xls-file "by hand".

A highly-functional and free spreadsheet application can be obtained as part of the cross-platform `OpenOffice` suite. It works for the above required conversion. Go to `"http://www.openoffice.org/"`.

It is regrettable that this importer does not take the xls-file directly as input. However, because of the moving-target nature of spreadsheet file formats, there is limited support for direct import of these formats into `R`. Rather than add to the dependency requirements of emphphyloseq and the relative support of these xls-support packages, it seems more efficient to choose an arbitrary delimited text format, and focus on the data structure in the PyroTagger output. This will be easier to support in the long-run.

Value

An `otuTax` object containing both the otu_table and TaxonomyTable data components, parsed from the pyrotagger output.

References

`http://pyrotagger.jgi-psf.org/`

Examples

```
## New_otuTaxObject <- import_pyrotagger_tab(pyrotagger_tab_file)
```

`import_qiime` *Import function to read the now legacy-format QIIME OTU table.*

Description

QIIME produces several files that can be directly imported by the `phyloseq-package`. Originally, QIIME produced its own custom format table that contained both OTU-abundance and taxonomic identity information. This function is still included in phyloseq mainly to accommodate these now-outdated files. Recent versions of QIIME store output in the biom-format, an emerging file format standard for microbiome data. If your data is in the biom-format, if it ends with a `.biom` file name extension, then you should use the `import_biom` function instead.

Usage

```
import_qiime(otufilename = NULL, mapfilename = NULL,
  treefilename = NULL, refseqfilename = NULL,
  refseqFunction = readDNAStringSet, refseqArgs = NULL,
  parseFunction = parse_taxonomy_qiime, verbose = TRUE, ...)
```

Arguments

`otufilename`	(Optional). A character string indicating the file location of the OTU file. The combined OTU abundance and taxonomic identification file, tab-delimited, as produced by QIIME under default output settings. Default value is `NULL`.
`mapfilename`	(Optional). The QIIME map file is required for processing barcoded primers in QIIME as well as some of the post-clustering analysis. This is a required input file for running QIIME. Its strict formatting specification should be followed for correct parsing by this function. Default value is `NULL`.
`treefilename`	(Optional). Default value is `NULL`. A file representing a phylogenetic tree or a `phylo` object. Files can be NEXUS or Newick format. See `read_tree` for more details. Also, if using a recent release of the GreenGenes database tree, try the `read_tree_greengenes` function – this should solve some issues specific to importing that tree. If provided, the tree should have the same OTUs/tip-labels as the OTUs in the other files. Any taxa or samples missing in one of the files is removed from all. As an example from the QIIME pipeline, this tree would be a tree of the representative 16S rRNA sequences from each OTU cluster, with the number of leaves/tips equal to the number of taxa/species/OTUs, or the complete reference database tree that contains the OTU identifiers of every OTU in your abundance table. Note that this argument can be a tree object (`phylo`-class) for cases where the tree has been — or needs to be — imported separately, as in the case of the GreenGenes tree mentioned earlier (coderead_tree_greengenes).
`refseqfilename`	(Optional). Default `NULL`. The file path of the biological sequence file that contains at a minimum a sequence for each OTU in the dataset. Alternatively, you may provide an already-imported `XStringSet` object that satisfies this condition. In either case, the `names` of each OTU need to match exactly the `taxa_names` of the other components of your data. If this is not the case, for example if the data file is a FASTA format but contains additional information after the OTU name in each sequence header, then some additional parsing is necessary, which you can either perform separately before calling this function, or describe explicitly in a custom function provided in the (next) argument, `refseqFunction`.

	Note that the `XStringSet` class can represent any arbitrary sequence, including user-defined subclasses, but is most-often used to represent RNA, DNA, or amino acid sequences. The only constraint is that this special list of sequences has exactly one named element for each OTU in the dataset.
`refseqFunction`	(Optional). Default is `readDNAStringSet`, which expects to read a fasta-formatted DNA sequence file. If your reference sequences for each OTU are amino acid, RNA, or something else, then you will need to specify a different function here. This is the function used to read the file connection provided as the the previous argument, `refseqfilename`. This argument is ignored if `refseqfilename` is already a `XStringSet` class.
`refseqArgs`	(Optional). Default `NULL`. Additional arguments to `refseqFunction`. See `XStringSet-io` for details about additional arguments to the standard read functions in the Biostrings package.
`parseFunction`	(Optional). An optional custom function for parsing the character string that contains the taxonomic assignment of each OTU. The default parsing function is `parse_taxonomy_qiime`, specialized for splitting the `";"`-delimited strings and also attempting to interpret greengenes prefixes, if any, as that is a common format of the taxonomy string produced by QIIME.
`verbose`	(Optional). A `logical`. Default is `TRUE`. Should progresss messages be `cat`ted to standard out?
`...`	Additional arguments passed to `read_tree`

Details

Other related files include the mapping-file that typically stores sample covariates, converted naturally to the `sample_data-class` component data type in the phyloseq-package. QIIME may also produce a phylogenetic tree with a tip for each OTU, which can also be imported specified here or imported separately using `read_tree`.

See `"http://www.qiime.org/"` for details on using QIIME. While there are many complex dependencies, QIIME can be downloaded as a pre-installed linux virtual machine that runs “off the shelf”.

The different files useful for import to *phyloseq* are not collocated in a typical run of the QIIME pipeline. See the main *phyloseq* vignette for an example of where ot find the relevant files in the output directory.

Value

A `phyloseq-class` object.

References

`http://qiime.org/`

“QIIME allows analysis of high-throughput community sequencing data.” J Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D Bushman, Elizabeth K Costello, Noah Fierer, Antonio Gonzalez Pena, Julia K Goodrich, Jeffrey I Gordon, Gavin A Huttley, Scott T Kelley, Dan Knights, Jeremy E Koenig, Ruth E Ley, Catherine A Lozupone, Daniel McDonald, Brian D Muegge, Meg Pirrung, Jens Reeder, Joel R Sevinsky, Peter J Turnbaugh, William A Walters, Jeremy Widmann, Tanya Yatsunenko, Jesse Zaneveld and Rob Knight; Nature Methods, 2010; doi:10.1038/nmeth.f.303

See Also

phyloseq

merge_phyloseq

read_tree

read_tree_greengenes

XStringSet-io

Examples

```
otufile <- system.file("extdata", "GP_otu_table_rand_short.txt.gz", package="phyloseq")
mapfile <- system.file("extdata", "master_map.txt", package="phyloseq")
trefile <- system.file("extdata", "GP_tree_rand_short.newick.gz", package="phyloseq")
import_qiime(otufile, mapfile, trefile)
```

`import_qiime_otu_tax` *Import now legacy-format QIIME OTU table as a list of two matrices.*

Description

Now a legacy-format, older versions of QIIME produced an OTU file that typically contains both OTU-abundance and taxonomic identity information in a tab-delimted table. If your file ends with the extension `.biom`, or if you happen to know that it is a biom-format file, or if you used default settings in a version of QIIME of `1.7` or greater, then YOU SHOULD USE THE BIOM-IMPORT FUNCTION instead, `import_biom`.

Usage

```
import_qiime_otu_tax(file, parseFunction = parse_taxonomy_qiime,
  verbose = TRUE, parallel = FALSE)
```

Arguments

`file`	(Required). The path to the qiime-formatted file you want to import into R. Can be compressed (e.g. `.gz`, etc.), though the details may be OS-specific. That is, Windows-beware.
`parseFunction`	(Optional). An optional custom function for parsing the character string that contains the taxonomic assignment of each OTU. The default parsing function is `parse_taxonomy_qiime`, specialized for splitting the `";"`-delimited strings and also attempting to interpret greengenes prefixes, if any, as that is a common format of the taxonomy string produced by QIIME.
`verbose`	(Optional). A `logical`. Default is `TRUE`. Should progresss messages be `cat`ted to standard out?
`parallel`	(Optional). Logical. Should the parsing be performed in parallel?. Default is `FALSE`. Only a few steps are actually parallelized, and for most datasets it will actually be faster and more efficient to keep this set to `FALSE`. Also, to get any benefit at all, you will need to register a parallel "backend" through one of the backend packages supported by the `foreach-package`.

Details

This function uses chunking to perform both the reading and parsing in blocks of optional size, thus constrain the peak memory usage. feature should make this importer accessible to machines with modest memory, but with the caveat that the full numeric matrix must be a manageable size at the end, too. In principle, the final tables will be large, but much more efficiently represented than the character-stored numbers. If total memory for storing the numeric matrix becomes problematic, a switch to a sparse matrix representation of the abundance – which is typically well-suited to this data – might provide a solution.

Value

A list of two matrices. `$otutab` contains the OTU Table as a numeric matrix, while `$taxtab` contains a character matrix of the taxonomy assignments.

See Also

`import`

`merge_phyloseq`

`phyloseq`

`import_qiime`

`read_tree`

`read_tree_greengenes`

`import_env_file`

Examples

```
otufile <- system.file("extdata", "GP_otu_table_rand_short.txt.gz", package="phyloseq")
import_qiime_otu_tax(otufile)
```

`import_qiime_sample_data`

Import just `sample_data` *file from QIIME pipeline.*

Description

QIIME produces several files that can be analyzed in the phyloseq-package, This includes the map-file, which is an important *input* to QIIME that can also indicate sample covariates. It is converted naturally to the sample_data component data type in phyloseq-package, based on the R data.frame.

Usage

```
import_qiime_sample_data(mapfilename)
```

Arguments

`mapfilename`	(Required). A character string or connection. That is, any suitable `file` argument to the `read.table` function. The name of the QIIME map file required for processing pyrosequencing tags in QIIME as well as some of the post-clustering analysis. This is a required input file for running QIIME. Its strict formatting specification is expected by this function, do not attempt to modify it manually once it has worked properly in QIIME.

Details

See `import_qiime` for more information about QIIME. It is also the suggested function for importing QIIME-produced data files.

Value

A `sample_data` object.

See Also

`import`

`merge_phyloseq`

`phyloseq`

`import_qiime`

`import_qiime_otu_tax`

`import_env_file`

Examples

```
mapfile <- system.file("extdata", "master_map.txt", package = "phyloseq")
import_qiime_sample_data(mapfile)
```

`import_RDP_cluster` *Import RDP cluster file and return otu_table (abundance table).*

Description

The RDP cluster pipeline (specifically, the output of the complete linkage clustering step) has no formal documentation for the `".clust"` file or its apparent sequence naming convention.

Usage

```
import_RDP_cluster(RDP_cluster_file)
```

Arguments

`RDP_cluster_file`	A character string. The name of the `".clust"` file produced by the the complete linkage clustering step of the RDP pipeline.

Details

`http://pyro.cme.msu.edu/index.jsp`

The cluster file itself contains the names of all sequences contained in input alignment. If the upstream barcode and aligment processing steps are also done with the RDP pipeline, then the sequence names follow a predictable naming convention wherein each sequence is named by its sample and sequence ID, separated by a `"_"` as delimiter:

`"sampleName_sequenceIDnumber"`

This import function assumes that the sequence names in the cluster file follow this convention, and that the sample name does not contain any `"_"`. It is unlikely to work if this is not the case. It is likely to work if you used the upstream steps in the RDP pipeline to process your raw (barcoded, untrimmed) fasta/fastq data.

This function first loops through the `".clust"` file and collects all of the sample names that appear. It secondly loops through each OTU (`"cluster"`; each row of the cluster file) and sums the number of sequences (reads) from each sample. The resulting abundance table of OTU-by-sample is trivially coerced to an `otu_table` object, and returned.

Value

An `otu_table` object parsed from the `".clust"` file.

References

`http://pyro.cme.msu.edu/index.jsp`

`import_RDP_otu` *Import new RDP OTU-table format*

Description

Recently updated tools on RDP Pyro site make it easier to import Pyrosequencing output into R. The modified tool “Cluster To R Formatter” can take a cluster file (generated from RDP Clustering tools) to create a community data matrix file for distance cutoff range you are interested in. The resulting output file is a tab-delimited file containing the number of sequences for each sample for each OTU. The OTU header naming convention is `"OTU_"` followed by the OTU number in the cluster file. It pads “0”s to make the OTU header easy to sort. The OTU numbers are not necessarily in order.

Usage

```
import_RDP_otu(otufile)
```

Arguments

`otufile`	(Optional). A character string indicating the file location of the OTU file, produced/exported according to the instructions above.

Value

A `otu_table-class` object.

See Also

An alternative “cluster” file importer for RDP results: `import_RDP_cluster`

The main RDP-pyrosequencing website `http://pyro.cme.msu.edu/index.jsp`

Examples

```
otufile <- system.file("extdata", "rformat_dist_0.03.txt.gz", package="phyloseq")
### the gzipped file is automatically recognized, and read using R-connections
ex_otu  <- import_RDP_otu(otufile)
class(ex_otu)
ntaxa(ex_otu)
nsamples(ex_otu)
sample_sums(ex_otu)
head(t(ex_otu))
```

`import_uparse` *Import* R*hrefhttp://www.drive5.com/usearch/manual/opt_uparseout.htmlUPARSE file format*

Description

UPARSE is an algorithm for OTU-clustering implemented within usearch. At last check, the UPARSE algortihm was accessed via the `-cluster_otu` option flag. For details about installing and running usearch, please refer to the usearch website. For details about the output format, please refer to the uparse format definition.

Usage

```
import_uparse(upFile, omitChimeras = TRUE, countTable = TRUE,
  OTUtable = TRUE, verbose = TRUE)
```

Arguments

`upFile`	(Required). A file location character string or `connection` corresponding to the file that contains the UPARSE output table. This is passed directly to `fread`. Please see its `file` argument documentation for further links and details.
`omitChimeras`	(Optional). `logical(1)`. Default is `TRUE`. Whether to omit entries that correspond to sequences/OTUs that were identified as chimeras.
`countTable`	(Optional). `logical(1)`. Default is `TRUE`. Whether to return the result as a wide-format table with dimensions OTU-by-sample, or to leave the table in its original sparse long-format that might be more suitable for certain `data.table` operations. If `TRUE`, entries corresponding to the same sample and OTU have their counts summed.
`OTUtable`	(Optional). `logical(1)`. Default is `TRUE`. Whether to coerce the result to `otu_table` format, or leave it as a `data.table` format. The former is appropriate for most `phyloseq` operations, the latter is useful for a lot of custom operations and custom `ggplot2` graphics calls.
`verbose`	(Optional). A `logical`. Default is `TRUE`. Should progresss messages be `cat`ted to standard out?

Details

Because UPARSE is an external (non-R) application, there is no direct way to continuously check that these suggested arguments and file formats will remain in their current state. If there is a problem, please verify your version of usearch, create a small reproducible example of the problem, and post it as an issue on the phyloseq issues tracker.

See Also

`import_usearch_uc`

Examples

```
###
```

`import_usearch_uc` *Import usearch table format (`.uc`) to OTU table*

Description

UPARSE is an algorithm for OTU-clustering implemented within usearch. At last check, the UPARSE algortihm was accessed via the `-cluster_otu` option flag. For details about installing and running usearch, please refer to the usearch website. For details about the output format, please refer to the uc format definition. This importer is intended to read a particular table format output that is generated by usearch, its so-called "cluster format", a file format that is often given the `.uc` extension in usearch documentation.

Usage

```
import_usearch_uc(ucfile, colRead = 9, colOTU = 10,
  readDelimiter = "_", verbose = TRUE)
```

Arguments

`ucfile`	(Required). A file location character string or `connection` corresponding to the file that contains the usearch output table. This is passed directly to `read.table`. Please see its `file` argument documentation for further links and details.
`colRead`	(Optional). Numeric. The column index in the uc-table file that holds the read IDs. The default column index is `9`.
`colOTU`	(Optional). Numeric. The column index in the uc-table file that holds OTU IDs. The default column index is `10`.
`readDelimiter`	(Optional). An R `regex` as a character string. This should be the delimiter that separates the sample ID from the original ID in the demultiplexed read ID of your sequence file. The default is plain underscore, which in this `regex` context is `"_"`.
`verbose`	(Optional). A `logical`. Default is `TRUE`. Should progresss messages be `cat`ted to standard out?

Details

Because usearch is an external (non-R) application, there is no direct way to continuously check that these suggested arguments and file formats will remain in their current state. If there is a problem, please verify your version of usearch, create a small reproducible example of the problem, and post it as an issue on the phyloseq issues tracker. The version of usearch upon which this import function was created is `7.0.109`. Hopefully later versions of usearch maintain this function and format, but the phyloseq team has no way to guarantee this, and so any feedback about this will help maintain future functionality. For instance, it is currently assumed that the 9th and 10th columns of the `.uc` table hold the read-label and OTU ID, respectively; and it is also assumed that the delimiter between sample-name and read in the read-name entries is a single `"_"`. If this is not true, you may have to update these parameters, or even modify the current implementation of this function.

Also note that there is now a UPARSE-specific output file format, uparseout, and it might make more sense to create and import that file for use in phyloseq. If so, you'll want to import using the `import_uparse()` function.

See Also

`import`

`import_biom`

`import_qiime`

Examples

```
usearchfile <- system.file("extdata", "usearch.uc", package="phyloseq")
import_usearch_uc(usearchfile)
```

`index_reorder` *Force index order of phyloseq objects*

Description

Force index order of phyloseq objects

Usage

```
index_reorder(ps, index_type)

## S4 method for signature 'phyloseq'
index_reorder(ps, index_type = "both")
```

Arguments

`ps`	(Required). A `phyloseq-class` instance.
`index_type`	(Optional). A character string specifying the indices to properly order. Supported values are `c("both", "taxa", "samples")`. Default is `"both"`, meaning samples and taxa indices will be checked/re-ordered.

Examples

```
## data("GlobalPatterns")
## GP = index_reorder(GlobalPatterns)
```

`intersect_taxa` *Returns the intersection of species and samples for the components of x*

Description

This function is used internally as part of the infrastructure to ensure that component data types in a phyloseq-object have exactly the same taxa/species. It relies heavily on the `Reduce` function to determine the strictly common species.

Usage

```
intersect_taxa(x)
```

Arguments

`x`	(Required). A `phyloseq-class` object that contains 2 or more components that in-turn describe species/taxa.

Value

Returns a character vector of only those species that are present in all species-describing components of `x`.

See Also

`Reduce`, `intersect`

Examples

```
#
## data(GlobalPatterns)
## head(intersect_taxa(GlobalPatterns), 10)
```

`JSD` *Calculate the Jensen-Shannon Divergence (distance)*

Description

This is a phyloseq-specific implementation of the Jensen-Shannon Divergence for comparing pairs of microbial communities (samples) in an experiment. The expectation is that you have many samples (say. more than two) and you want a distance matrix on which will perform further analysis. `JSD` is intended to be “wrapped” by the more general `distance` function in phyloseq, and it can be invoked using `"jsd"` as the argument to the `method` parameter of `distance`.

Usage

```
JSD(physeq)
```

Arguments

`physeq`	(Required). `phyloseq-class`. The phyloseq data on which to compute the pairwise sample distance matrix.

Details

One of the motivations for providing JSD in phyloseq was its recent use in the analysis of the `enterotype` dataset.

Value

An object of class "`dist`" suitable for certain ordination methods and other distance-based analyses. See `distance`.

Author(s)

Susan Holmes `<susan@stat.stanford.edu>`. Adapted for phyloseq by Paul J. McMurdie.

References

Jensen-Shannon Divergence and Hilbert space embedding. Bent Fuglede and Flemming Topsoe University of Copenhagen, Department of Mathematics `http://www.math.ku.dk/~topsoe/ISIT2004JSD.pdf`

See Also

`distance`

`enterotype`

`http://en.wikipedia.org/wiki/Jensen-Shannon_divergence`

Examples

```
# library(doParallel)  # Do this and next line only if you have multi-cores
# registerDoParallel(cores=6)
# data(enterotype)
# # ent.jsd <- JSD(enterotype, TRUE) # internal only
# ent.jsd <- distance(enterotype, "jsd", parallel=TRUE)
# ent.PCoA <- ordinate(enterotype, "PCoA", ent.jsd) # Perform principle coordinate analysis
# p <- plot_ordination(enterotype, ent.PCoA, color="Enterotype", shape="SeqTech")
# (p <- p + geom_point(size=5, alpha=0.5))
```

`make_network` *Make microbiome network (igraph)*

Description

A specialized function for creating a network representation of microbiomes, sample-wise or taxa-wise, based on a user-defined ecological distance and (potentially arbitrary) threshold. The graph is ultimately represented using the `igraph`-package.

Usage

```
make_network(physeq, type="samples", distance="jaccard", max.dist = 0.4,
    keep.isolates=FALSE, ...)
```

Arguments

`physeq`	(Required). Default `NULL`. A `phyloseq-class` object, or `otu_table-class` object, on which g is based. `phyloseq-class` recommended.
`type`	(Optional). Default `"samples"`. Whether the network should be samples or taxa/OTUs. Supported arguments are `"samples"`, `"taxa"`, where `"taxa"` indicates using the OTUs/taxaindices, whether they actually represent species or some other taxonomic rank. NOTE: not all distance methods are supported if `"taxa"` selected for type. For example, the UniFrac distance and DPCoA cannot be calculated for taxa-wise distances, because they use a taxa-wise tree as part of their calculation between samples, and there is no transpose-equivalent for this tree.
`distance`	(Optional). Default `"jaccard"`. Any supported argument to the `method` parameter of the `distance` function is supported here. Some distance methods, like `"unifrac"`, may take a non-trivial amount of time to calculate, in which case you probably want to calculate the distance matrix separately, save, and then provide it as the argument to `distance` instead. See below for alternatives). Alternatively, if you have already calculated the sample-wise distance object, the resulting `dist`-class object can be provided as `distance` instead (see examples). A third alternative is to provide a function that takes a sample-by-taxa matrix (typical vegan orientation) and returns a sample-wise distance matrix.
`max.dist`	(Optional). Default `0.4`. The maximum ecological distance (as defined by `distance`) allowed between two samples to still consider them “connected” by an edge in the graphical model.
`keep.isolates`	(Optional). Default `FALSE`. Logical. Whether to keep isolates (un-connected samples, not microbial isolates) in the graphical model that is returned. Default results in isolates being removed from the object.
`...`	(Optional). Additional parameters passed on to `distance`.

Value

A `igraph`-class object.

See Also

`plot_network`

Examples

```
# # Example plots with Enterotype Dataset
data(enterotype)
ig <- make_network(enterotype, max.dist=0.3)
plot_network(ig, enterotype, color="SeqTech", shape="Enterotype", line_weight=0.3, label=NULL)
#
ig1 <- make_network(enterotype, max.dist=0.2)
plot_network(ig1, enterotype, color="SeqTech", shape="Enterotype", line_weight=0.3, label=NULL)
#
# # Three methods of choosing/providing distance/distance-method
```

```
# Provide method name available to distance() function
ig <- make_network(enterotype, max.dist=0.3, distance="jaccard")
# Provide distance object, already computed
jaccdist <- distance(enterotype, "jaccard")
ih <- make_network(enterotype, max.dist=0.3, distance=jaccdist)
# Provide "custom" function.
ii <- make_network(enterotype, max.dist=0.3, distance=function(x){vegan::vegdist(x, "jaccard")})
# The have equal results:
all.equal(ig, ih)
all.equal(ig, ii)
#
# Try out making a trivial "network" of the 3-sample esophagus data,
# with weighted-UniFrac as distance
data(esophagus)
ij <- make_network(esophagus, "samples", "unifrac", weighted=TRUE)
```

`merge_phyloseq` *Merge arguments into one phyloseq object.*

Description

Takes a comma-separated list of phyloseq objects as arguments, and returns the most-comprehensive single phyloseq object possible.

Usage

```
merge_phyloseq(...)
```

Arguments

`...`	a comma-separated list of phyloseq objects.

Details

Higher-order objects can be created if arguments are appropriate component data types of different classes, and this should mirror the behavior of the `phyloseq` method, which is the suggested method if the goal is simply to create a higher-order phyloseq object from different data types (1 of each class) describing the same experiment.

By contrast, this method is intended for situations in which one wants to combine multiple higher-order objects, or multiple core component data objects (e.g. more than one `otu_table`) that should be combined into one object.

Merges are performed by first separating higher-order objects into a list of their component objects; then, merging any component objects of the same class into one object according to the behavior desribed in `merge_phyloseq_pair`; and finally, building back up a merged-object according to the constructor behavior of the `phyloseq` method. If the arguments contain only a single component type – several otu_table objects, for example – then a single merged object of that component type is returned.

Value

Merges are performed by first separating higher-order objects into a list of their component objects; then, merging any component objects of the same class into one object according to the behavior desribed in `merge_phyloseq_pair`; and finally, re-building a merged-object according to the constructor behavior of the `phyloseq` method. If the arguments contain only a single component type – several otu_table objects, for example – then a single merged object of the relevant component type is returned.

Merges between 2 or more tree objects are ultimately done using `consensus` from the ape package. This has the potential to limit somewhat the final data object, because trees don't merge with other trees in the same granular manner as data tables, and ultimately the species/taxa in higher-order phyloseq objects will be clipped to what is contained in the tree. If this an issue, the tree component should be ommitted from the argument list.

Examples

```
#
## # Make a random complex object
## OTU1 <- otu_table(matrix(sample(0:5,250,TRUE),25,10), taxa_are_rows=TRUE)
## tax1 <- tax_table(matrix("abc", 30, 8))
## map1 <- data.frame( matrix(sample(0:3,250,TRUE),25,10),
##   matrix(sample(c("a","b","c"),150,TRUE), 25, 6) )
## map1 <- sample_data(map1)
## exam1 <- phyloseq(OTU1, map1, tax1)
## x <- exam1
## x <- phyloseq(exam1)
## y <- tax_table(exam1)
## merge_phyloseq(x, y)
## merge_phyloseq(y, y, y, y)
```

`merge_phyloseq_pair` *Merge pair of phyloseq component data objects of the same class.*

Description

Internal S4 methods to combine pairs of objects of classes specified in the phyloseq package. These objects must be component data of the same type (class). This is mainly an internal method, provided to illustrate how merging is performed by the more general `merge_phyloseq` function.

Usage

```
merge_phyloseq_pair(x, y)

## S4 method for signature 'otu_table,otu_table'
merge_phyloseq_pair(x, y)

## S4 method for signature 'taxonomyTable,taxonomyTable'
merge_phyloseq_pair(x, y)

## S4 method for signature 'sample_data,sample_data'
merge_phyloseq_pair(x, y)
```

```
## S4 method for signature 'phylo,phylo'
merge_phyloseq_pair(x, y)

## S4 method for signature 'XStringSet,XStringSet'
merge_phyloseq_pair(x, y)
```

Arguments

x	A character vector of the species in object x that you want to keep – OR alternatively – a logical vector where the kept species are TRUE, and length is equal to the number of species in object x. If `species` is a named logical, the species retained is based on those names. Make sure they are compatible with the `taxa_names` of the object you are modifying (x).
y	Any phyloseq object.

Details

The `merge_phyloseq` function is recommended in general.

Special note: non-identical trees are merged using `consensus`.

Value

A single component data object that matches `x` and `y` arguments. The returned object will contain the union of the species and/or samples of each. If there is redundant information between a pair of arguments of the same class, the values in `x` are used by default. Abundance values are summed for `otu_table` objects for those elements that describe the same species and sample in `x` and `y`.

See Also

`merge_phyloseq` `merge_taxa`

Examples

```
#
## # merge two simulated otu_table objects.
## x  <- otu_table(matrix(sample(0:5,200,TRUE),20,10), taxa_are_rows=TRUE)
## y  <- otu_table(matrix(sample(0:5,300,TRUE),30,10), taxa_are_rows=FALSE)
## xy <- merge_phyloseq_pair(x, y)
## yx <- merge_phyloseq_pair(y, x)
## # merge two simulated tax_table objects
## x <- tax_table(matrix("abc", 20, 6))
## y <- tax_table(matrix("def", 30, 8))
## xy <- merge_phyloseq_pair(x, y)
## # merge two simulated sample_data objects
## x <- data.frame( matrix(sample(0:3,250,TRUE),25,10),
##   matrix(sample(c("a","b","c"),150,TRUE),25,6) )
## x <- sample_data(x)
## y <- data.frame( matrix(sample(4:6,200,TRUE),20,10),
##   matrix(sample(c("d","e","f"),120,TRUE),20,8) )
## y <- sample_data(y)
## merge_phyloseq_pair(x, y)
## data.frame(merge_phyloseq_pair(x, y))
## data.frame(merge_phyloseq_pair(y, x))
```

merge_samples *Merge samples based on a sample variable or factor.*

Description

The purpose of this method is to merge/agglomerate the sample indices of a phyloseq object according to a categorical variable contained in a sample_data or a provided factor.

Usage

```
merge_samples(x, group, fun=mean)

## S4 method for signature 'sample_data'
merge_samples(x, group, fun = mean)

## S4 method for signature 'otu_table'
merge_samples(x, group)

## S4 method for signature 'phyloseq'
merge_samples(x, group, fun = mean)
```

Arguments

`x`	(Required). An instance of a phyloseq class that has sample indices. This includes `sample_data-class`, `otu_table-class`, and `phyloseq-class`.
`group`	(Required). Either the a single character string matching a variable name in the corresponding sample_data of `x`, or a factor with the same length as the number of samples in `x`.
`fun`	(Optional). The function that will be used to merge the values that correspond to the same group for each variable. It must take a numeric vector as first argument and return a single value. Default is `mean`. Note that this is (currently) ignored for the otu_table, where the equivalent function is `sum`, but evaluated via `rowsum` for efficiency.

Details

NOTE: (`phylo`) trees and `taxonomyTable-class` are not modified by this function, but returned in the output object as-is.

Value

A phyloseq object that has had its sample indices merged according to the factor indicated by the `group` argument. The output class matches `x`.

See Also

`merge_taxa`, codemerge_phyloseq

Examples

```
#
data(GlobalPatterns)
GP = GlobalPatterns
mergedGP = merge_samples(GlobalPatterns, "SampleType")
SD = merge_samples(sample_data(GlobalPatterns), "SampleType")
print(SD)
print(mergedGP)
sample_names(GlobalPatterns)
sample_names(mergedGP)
identical(SD, sample_data(mergedGP))
# The OTU abundances of merged samples are summed
# Let's investigate this ourselves looking at just the top10 most abundance OTUs...
OTUnames10 = names(sort(taxa_sums(GP), TRUE)[1:10])
GP10  = prune_taxa(OTUnames10,  GP)
mGP10 = prune_taxa(OTUnames10, mergedGP)
ocean_samples = sample_names(subset(sample_data(GP), SampleType=="Ocean"))
print(ocean_samples)
otu_table(GP10)[, ocean_samples]
rowSums(otu_table(GP10)[, ocean_samples])
otu_table(mGP10)["Ocean", ]
```

`merge_taxa` *Merge a subset of the species in* `x` *into one species/taxa/OTU.*

Description

Takes as input an object that describes species/taxa (e.g. `phyloseq-class`, `otu_table-class`, `phylo-class`, `taxonomyTable-class`), as well as a vector of species that should be merged. It is intended to be able to operate at a low-level such that related methods, such as `tip_glom` and `tax_glom` can both reliably call `merge_taxa` for their respective purposes.

Usage

```
merge_taxa(x, eqtaxa, archetype=1)

## S4 method for signature 'phyloseq'
merge_taxa(x, eqtaxa,
  archetype = eqtaxa[which.max(taxa_sums(x)[eqtaxa])])

## S4 method for signature 'sample_data'
merge_taxa(x, eqtaxa, archetype = 1L)

## S4 method for signature 'otu_table'
merge_taxa(x, eqtaxa,
  archetype = eqtaxa[which.max(taxa_sums(x)[eqtaxa])])

## S4 method for signature 'phylo'
merge_taxa(x, eqtaxa, archetype = 1L)

## S4 method for signature 'XStringSet'
merge_taxa(x, eqtaxa, archetype = 1L)
```

```
## S4 method for signature 'taxonomyTable'
merge_taxa(x, eqtaxa, archetype = 1L)
```

Arguments

`x`	(Required). An object that describes species (taxa). This includes `phyloseq-class`, `otu_table-class`, `taxonomyTable-class`, `phylo`.
`eqtaxa`	(Required). The species names, or indices, that should be merged together. If `length(eqtaxa)` < 2, then the object `x` will be returned safely unchanged.
`archetype`	(Optional). A single-length numeric or character. The index of `eqtaxa`, or OTU ID, indicating the species that should be kept to represent the summed/merged group of species/taxa/OTUs. The default is to use the OTU with the largest count total if counts are available, or to use `1` (the first OTU in `eqtaxa`) otherwise. If `archetype` is not a valid index or index-name in `eqtaxa`, the first will be used, and the value in archetype will be used as the index-name for the new species.

Value

The object, `x`, in its original class, but with the specified species merged into one entry in all relevant components.

See Also

`tip_glom`, `tax_glom`, `merge_phyloseq`, `merge_samples`

Examples

```
#
data(esophagus)
tree <- phy_tree(esophagus)
otu  <- otu_table(esophagus)
otutree0 <- phyloseq(otu, tree)
# plot_tree(otutree0)
otutree1 <- merge_taxa(otutree0, 1:8, 2)
# plot_tree(esophagus, ladderize="left")
```

`metaMDS` *S3 class placeholder definition (list) for metaMDS*

Description

The ape package does export a version of its `metaMDS`-class, partly because it is not really defined formally anywhere. Instead, it is an S3 class extended from the base class, `list` – this is a very common and easy approach – and proper behavior of any method taking an instance of this class requires exact naming conventions for element names of the list components. The phyloseq package does not provide any validity checks that a given phylo instance is valid (conforms to the conventions in the ape package)... yet. If problems arise, this might be considered, and they could be defined judiciously and within phyloseq.

Usage

```
metaMDS
```

Format

An object of class `metaMDS` of length 0.

See Also

`metaMDS`

`microbio_me_qiime` *Import microbio.me/qiime (QIIME-DB) data package*

Description

Originally, this function was for accessing microbiome datasets from the microbio.me/qiime public repository from within R. As you can see by clicking on the above link, the QIIME-DB sever is down indefinitely. However, this function will remain supported here in case the FTP server goes back up, and also for phyloseq users that have downloaded one or more data packages prior to the server going down.

Usage

```
microbio_me_qiime(zipftp, ext = ".zip",
  parsef = parse_taxonomy_greengenes, ...)
```

Arguments

`zipftp`	(Required). A character string that is the full URL path to a zipped file that follows the file naming conventions used by microbio.me/qiime. Alternatively, you can simply provide the study number as a single `integer` or other single-length vector that can be `coerce`d to an integer; this function will complete the remainder of the ftp URL hosted at microbio.me/qiime. For example, instead of the full URL string, `"ftp://thebeast.colorado.edu/pub/QIIME_DB_Public_Studies/study_494_spli` you could simply provide `494` or `"494"` as the first (‘zipftp‘) argument.
`ext`	(Optional). A `character` string of the expected file extension, which also indicates the compression type, if `zipftp` is a study number instead of the full path. Note that this argument has no effect if `zipftp` is the full path, in which case the file extension is read directly from the downloaded file.
`parsef`	(Optional). The type of taxonomic parsing to use for the OTU taxonomic classification, in the `.biom` file, if present. This is passed on to `import_biom`, but unlike that function the default parsing function is `parse_taxonomy_greengenes`, rather than `parse_taxonomy_default`, because we know ahead of time that most (or all?) of the taxonomic classifications in the `microbio.me/qiime` repository will be based on GreenGenes.
`...`	(Optional, for advanced users). Additional arguments passed to `download.file`. This is mainly for non-standard links to resources (in this case, a zipped file) that are not being hosted by microbio.me/qiime. If you are using a FTP address or study number from their servers, then you shouldn't need to provide any additional arguments.

Value

A `phyloseq-class` object if possible, a component if only a component could be imported, or `NULL` if nothing could be imported after unzipping the file. Keep in mind there is a specific naming-convention that is expected based on the current state of the microbio.me/qiime servers. Several helpful messages are `cat`ted to standard out to help let you know the ongoing status of the current download and import process.

See Also

See `download.file` and `url` for details about URL formats – including local file addresses – that might work here.

`import_biom`

`import_qiime`

Examples

```
# This should return TRUE on your system if you have internet turned on
# and a standard R installation. Indicates whether this is likely to
# work on your system for a URL or local file, respectively.
capabilities("http/ftp"); capabilities("fifo")
# A working example with a local example file included in phyloseq
zipfile = "study_816_split_library_seqs_and_mapping.zip"
zipfile = system.file("extdata", zipfile, package="phyloseq")
tarfile = "study_816_split_library_seqs_and_mapping.tar.gz"
tarfile = system.file("extdata", tarfile, package="phyloseq")
tarps = microbio_me_qiime(tarfile)
zipps = microbio_me_qiime(zipfile)
identical(tarps, zipps)
tarps; zipps
plot_heatmap(tarps)
# An example that used to work, before the QIIME-DB server was turned off by its host.
# # Smokers dataset
# smokezip = "ftp://thebeast.colorado.edu/pub/QIIME_DB_Public_Studies/study_524_split_library_seqs_and_mapp
# smokers1 = microbio_me_qiime(smokezip)
# # Alternatively, just use the study number
# smokers2 = microbio_me_qiime(524)
# identical(smokers1, smokers2)
```

`mt`

Multiple testing of taxa abundance according to sample categories/classes

Description

Please note that it is up to you to perform any necessary normalizing / standardizing transformations prior to these tests. See for instance `transform_sample_counts`.

Usage

```
mt(physeq, classlabel, minPmaxT = "minP", method = "fdr", ...)

## S4 method for signature 'phyloseq,ANY'
```

```
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)

## S4 method for signature 'otu_table,integer'
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)

## S4 method for signature 'otu_table,numeric'
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)

## S4 method for signature 'otu_table,logical'
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)

## S4 method for signature 'otu_table,character'
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)

## S4 method for signature 'otu_table,factor'
mt(physeq, classlabel, minPmaxT = "minP",
  method = "fdr", ...)
```

Arguments

`physeq`	(Required). `otu_table-class` or `phyloseq-class`. In this multiple testing framework, different taxa correspond to variables (hypotheses), and samples to observations.
`classlabel`	(Required). A single character index of the sample-variable in the `sample_data` of physeq that will be used for multiple testing. Alternatively, `classlabel` can be a custom integer (or numeric coercable to an integer), character, or factor with length equal to `nsamples(physeq)`. NOTE: the default test applied to each taxa is a two-sample two-sided `t.test`, WHICH WILL FAIL with an error if you provide a data variable (or custom vector) that contains MORE THAN TWO classes. One alternative to consider is an F-test, by specifying `test="f"` as an additional argument. See the first example below, and/or further documentation of `mt.maxT` or `mt.minP` for other options and formal details.
`minPmaxT`	(Optional). Character string. `"mt.minP"` or `"mt.maxT"`. Default is to use `"mt.minP"`.
`method`	(Optional). Additional multiple-hypthesis correction methods. A character vector from the set `p.adjust.methods`. Default is `"fdr"`, for the Benjamini and Hochberg (1995) method to control False Discovery Rate (FDR). This argument is passed on to `p.adjust`, please see that documentation for more details.
`...`	(Optional). Additional arguments, forwarded to `mt.maxT` or `mt.minP`

Value

A dataframe with components specified in the documentation for `mt.maxT` or `mt.minP`, respectively.

See Also

`mt.maxT`

`mt.minP`

`p.adjust`

Examples

```
## # Simple example, testing genera that sig correlate with Enterotypes
data(enterotype)
# Filter samples that don't have Enterotype
x <- subset_samples(enterotype, !is.na(Enterotype))
# (the taxa are at the genera level in this dataset)
res = mt(x, "Enterotype", method="fdr", test="f", B=300)
head(res, 10)
## # Not surprisingly, Prevotella and Bacteroides top the list.
## # Different test, multiple-adjusted t-test, whether samples are ent-2 or not.
## mt(x, get_variable(x, "Enterotype")==2)
```

`nodeplotblank` *Function to avoid plotting node labels*

Description

Unlike, `nodeplotdefault` and `nodeplotboot`, this function does not return a function, but instead is provided directly to the `nodelabf` argument of `plot_tree` to ensure that node labels are not added to the graphic. Please note that you do not need to create or obtain the arguments to this function. Instead, you can provide this function directly to `plot_tree` and it will know what to do with it. Namely, use it to avoid plotting any node labels.

Usage

```
nodeplotblank(p, nodelabdf)
```

Arguments

`p`	(Required). The `plot_tree` graphic.
`nodelabdf`	(Required). The `data.frame` produced internally in `link{plot_tree}` to use as data for creating ggplot2-based tree graphics.

Value

The same input object, p, provided as input. Unmodified.

See Also

`nodeplotdefault`

`nodeplotboot`

`plot_tree`

Examples

```
data("esophagus")
plot_tree(esophagus)
plot_tree(esophagus, nodelabf=nodeplotblank)
```

`nodeplotboot` *Generates a function for labeling bootstrap values on a phylogenetic tree.*

Description

Is not a labeling function itself, but returns one. The returned function is specialized for labeling bootstrap values. Note that the function that is returned has two completely different arguments from the four listed here: the plot object already built by earlier steps in `plot_tree`, and the `data.frame` that contains the relevant plotting data for the nodes (especially `x, y, label`), respectively. See `nodeplotdefault` for a simpler example. The main purpose of this and `nodeplotdefault` is to provide a useful default function generator for arbitrary and bootstrap node labels, respectively, and also to act as examples of functions that can successfully interact with `plot_tree` to add node labels to the graphic.

Usage

```
nodeplotboot(highthresh=95L, lowcthresh=50L, size=2L, hjust=-0.2)
```

Arguments

`highthresh`	(Optional). A single integer between 0 and 100. Any bootstrap values above this threshold will be annotated as a black filled circle on the node, rather than the bootstrap percentage value itself.
`lowcthresh`	(Optional). A single integer between 0 and 100, less than `highthresh`. Any bootstrap values below this value will not be added to the graphic. Set to 0 or below to add all available values.
`size`	(Optional). Numeric. Should be positive. The size parameter used to control the text size of taxa labels. Default is `2`. These are ggplot2 sizes.
`hjust`	(Optional). The horizontal justification of the node labels. Default is `-0.2`.

Value

A function that can add a bootstrap-values layer to the tree graphic. The values are represented in two ways; either as black filled circles indicating very high-confidence nodes, or the bootstrap value itself printed in small text next to the node on the tree.

See Also

`nodeplotdefault`

`nodeplotblank`

`plot_tree`

Examples

```
nodeplotboot()
nodeplotboot(3, -0.4)
```

`nodeplotdefault` *Generates a default node-label function*

Description

Is not a labeling function itself, but returns one. The returned function is capable of adding whatever label is on a node. Note that the function that is returned has two completely different arguments to those listed here: the plot object already built by earlier steps in `plot_tree`, and the `data.frame` that contains the relevant plotting data for the nodes (especially `x, y, label`), respectively. See `nodeplotboot` for a more sophisticated example. The main purpose of this and `nodeplotboot` is to provide a useful default function generator for arbitrary and bootstrap node labels, respectively, and also to act as examples of functions that will successfully interact with `plot_tree` to add node labels to the graphic.

Usage

```
nodeplotdefault(size=2L, hjust=-0.2)
```

Arguments

`size`	(Optional). Numeric. Should be positive. The size parameter used to control the text size of taxa labels. Default is `2`. These are ggplot2 sizes.
`hjust`	(Optional). The horizontal justification of the node labels. Default is `-0.2`.

Value

A function that can add a node-label layer to a graphic.

See Also

`nodeplotboot`

`nodeplotblank`

`plot_tree`

Examples

```
nodeplotdefault()
nodeplotdefault(3, -0.4)
```

`nsamples` *Get the number of samples.*

Description

Get the number of samples.

Usage

```
nsamples(physeq)

## S4 method for signature 'ANY'
nsamples(physeq)

## S4 method for signature 'phyloseq'
nsamples(physeq)

## S4 method for signature 'otu_table'
nsamples(physeq)

## S4 method for signature 'sample_data'
nsamples(physeq)
```

Arguments

`physeq`	A `phyloseq-class`, `sample_data`, or `otu_table-class`.

Value

An integer indicating the total number of samples.

See Also

`taxa_names`, `sample_names`, `ntaxa`

Examples

```
#
data("esophagus")
tree <- phy_tree(esophagus)
OTU1 <- otu_table(esophagus)
nsamples(OTU1)
physeq1 <- phyloseq(OTU1, tree)
nsamples(physeq1)
```

`ntaxa` *Get the number of taxa/species.*

Description

Get the number of taxa/species.

Usage

```
ntaxa(physeq)

## S4 method for signature 'ANY'
ntaxa(physeq)

## S4 method for signature 'phyloseq'
```

```
ntaxa(physeq)

## S4 method for signature 'otu_table'
ntaxa(physeq)

## S4 method for signature 'taxonomyTable'
ntaxa(physeq)

## S4 method for signature 'phylo'
ntaxa(physeq)

## S4 method for signature 'XStringSet'
ntaxa(physeq)
```

Arguments

`physeq`	`phyloseq-class`, `otu_table-class`, `taxonomyTable-class`, or `phylo`

Value

An integer indicating the number of taxa / species.

See Also

taxa_names

Examples

```
data("esophagus")
ntaxa(esophagus)
phy_tree(esophagus)
ntaxa(phy_tree(esophagus))
```

`ordinate` *Perform an ordination on phyloseq data*

Description

This function wraps several commonly-used ordination methods. The type of ordination depends upon the argument to `method`. Try `ordinate("help")` or `ordinate("list")` for the currently supported method options.

Usage

```
ordinate(physeq, method = "DCA", distance = "bray", formula = NULL,
  ...)
```

Arguments

`physeq`	(Required). Phylogenetic sequencing data (`phyloseq-class`). The data on which you want to perform the ordination. In general, these methods will be based in some fashion on the abundance table ultimately stored as a contingency matrix (`otu_table-class`). If you're able to import data into `phyloseq-class` format, than you don't need to worry, as an `otu_table` is a required component of this class. In addition, some ordination methods require additional data, like a constraining variable or phylogenetic tree. If that is the case, the relevant data should be included in `physeq` prior to running. Integrating the data in this way also results in these different data components being checked for validity and completeness by the method.
`method`	(Optional). A character string. Default is `"DCA"`. Currently supported method options are: `c("DCA", "CCA", "RDA", "CAP", "DPCoA", "NMDS", "MDS", "PCoA")` **DCA** Performs detrended correspondence analysis using`decorana` **CCA** Performs correspondence analysis, or optionally, constrained correspondence analysis (a.k.a. canonical correspondence analysis), via `cca` **RDA** Performs redundancy analysis, or optionally principal components analysis, via `rda` **CAP** [Partial] Constrained Analysis of Principal Coordinates or distance-based RDA, via `capscale`. See `capscale.phyloseq` for more details. In particular, a `formula` argument must be provided. **DPCoA** Performs Double Principle Coordinate Analysis using a (corrected, if necessary) phylogenetic/patristic distance between species. The calculation is performed by `DPCoA`(), which ultimately uses `dpcoa` after making the appropriate accessions/corrections of the data. **NMDS** Performs Non-metric MultiDimenstional Scaling of a sample-wise ecological distance matrix onto a user-specified number of axes, `k`. By default, `k=2`, but this can be modified as a supplementary argument. This method is ultimately carried out by `metaMDS` after the appropriate accessions and distance calculations. Because `metaMDS` includes its own distance calculation wrappers to `vegdist`, and these provide additional functionality in the form of species scores, `ordinate` will pass-on the `distance` argument to `metaMDS` if it is among the supported `vegdist` methods. However, all distance methods supported by `distance` are supported here, including `"unifrac"` (the default) and `"DPCoA"`. **MDS/PCoA** Performs principal coordinate analysis (also called principle coordinate decomposition, multidimensional scaling (MDS), or classical scaling) of a distance matrix (Gower 1966), including two correction methods for negative eigenvalues. See `pcoa` for further details.
`distance`	(Optional). A character string. Default is `"bray"`. The name of a supported `distance` method; or, alternatively, a pre-computed `dist`-class object. This argument is only utilized if a distance matrix is required by the ordination method specified by the `method` argument (above). Any supported `distance` methods are supported arguments to `distance` here. See `distance` for more details, examples.
`formula`	(Optional). A model `formula`. Only relevant for certain ordination methods. The left hand side is ignored, defined by the `physeq` and `distance` arguemnts. The right hand side gives the constraining variables, and conditioning variables

can be given within a special function `Condition`. See `cca` or `capscale` for examples/details.

`...` (Optional). Additional arguments to supporting functions. For example, the additional argument `weighted=TRUE` would be passed on to `UniFrac` if `"unifrac"` were chosen as the `distance` option and `"MDS"` as the ordination `method` option. Alternatively, if `"DCA"` were chosen as the ordination `method` option, additional arguments would be passed on to the relevant ordination function, `decorana`, for example.

Value

An ordination object. The specific class of the returned object depends upon the ordination method, as well as the function/package that is called internally to perform it. As a general rule, any of the ordination classes returned by this function will be recognized by downstream tools in the `phyloseq` package, for example the ordination plotting function, `plot_ordination`.

See Also

The plot_ordination Tutorial

Related component ordination functions described within phyloseq:

`DPCoA`

Described/provided by other packages:

`cca`/`rda`, `decorana`, `metaMDS`, `pcoa`, `capscale`

NMDS and MDS/PCoA both operate on distance matrices, typically based on some pairwise comparison of the microbiomes in an experiment/project. There are a number of common methods to use to calculate these pairwise distances, and the most convenient function (from a `phyloseq` point of view) for calculating these distance matrices is the

`distance`

function. It can be thought of as a distance / dissimilarity-index companion function for `ordinate`, and indeed the distance options provided to `ordinate` are often simply passed on to `distance`.

A good quick summary of ordination is provided in the introductory vignette for vegan:

vegan introductory vignette

The following `R` task views are also useful for understanding the available tools in `R`:

Analysis of Ecological and Environmental Data

Multivariate Statistics

Examples

```
# See http://joey711.github.io/phyloseq/plot_ordination-examples
# for many more examples.
# plot_ordination(GP, ordinate(GP, "DCA"), "samples", color="SampleType")
```

`otu_table` *Build or access the otu_table.*

Description

This is the suggested method for both constructing and accessing Operational Taxonomic Unit (OTU) abundance (`otu_table-class`) objects. When the first argument is a matrix, otu_table() will attempt to create and return an otu_table-class object, which further depends on whether or not `taxa_are_rows` is provided as an additional argument. Alternatively, if the first argument is an experiment-level (`phyloseq-class`) object, then the corresponding `otu_table` is returned.

Usage

```
otu_table(object, taxa_are_rows, errorIfNULL=TRUE)

## S4 method for signature 'phyloseq'
otu_table(object, errorIfNULL = TRUE)

## S4 method for signature 'otu_table'
otu_table(object, errorIfNULL = TRUE)

## S4 method for signature 'matrix'
otu_table(object, taxa_are_rows)

## S4 method for signature 'data.frame'
otu_table(object, taxa_are_rows)

## S4 method for signature 'ANY'
otu_table(object, errorIfNULL = TRUE)
```

Arguments

`object`	(Required). An integer matrix, `otu_table-class`, or `phyloseq-class`.
`taxa_are_rows`	(Conditionally optional). Logical; of length 1. Ignored unless `object` is a matrix, in which case it is is required.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`. Ignored if `object` argument is a matrix (constructor invoked instead).

Value

An `otu_table-class` object.

See Also

`phy_tree`, `sample_data`, `tax_table` `phyloseq`, `merge_phyloseq`

Examples

```
#
# data(GlobalPatterns)
# otu_table(GlobalPatterns)
```

`otu_table-class` *The S4 class for storing taxa-abundance information.*

Description

Because orientation of these tables can vary by method, the orientation is defined explicitly in the `taxa_are_rows` slot (a logical). The `otu_table` class inherits the `matrix` class to store abundance values. Various standard subset and assignment nomenclature has been extended to apply to the `otu_table` class, including square-bracket, `t`, etc.

Details

taxa_are_rows A single logical specifying the orientation of the abundance table.

.Data This slot is inherited from the `matrix` class.

`otu_table<-` *Assign a new OTU Table to* `x`

Description

Assign a new OTU Table to `x`

Usage

```
otu_table(x) <- value

## S4 replacement method for signature 'phyloseq,otu_table'
otu_table(x) <- value

## S4 replacement method for signature 'otu_table,otu_table'
otu_table(x) <- value

## S4 replacement method for signature 'phyloseq,phyloseq'
otu_table(x) <- value
```

Arguments

`x`	(Required). `phyloseq-class`
`value`	(Required). `otu_table-class` or `phyloseq-class`.

Examples

```
# data(GlobalPatterns)
# # An example of pruning to just the first 100 taxa in GlobalPatterns.
# ex2a <- prune_taxa(taxa_names(GlobalPatterns)[1:100], GlobalPatterns)
# # The following 3 lines produces an ex2b that is equal to ex2a
# ex2b <- GlobalPatterns
# OTU <- otu_table(GlobalPatterns)[1:100, ]
# otu_table(ex2b) <- OTU
# identical(ex2a, ex2b)
```

```
# print(ex2b)
# # Relace otu_table by implying the component in context.
# ex2c <- GlobalPatterns
# otu_table(ex2c) <- ex2b
# identical(ex2a, ex2c)
```

`parse_taxonomy_default` *Parse elements of a taxonomy vector*

Description

These are provided as both example and default functions for parsing a character vector of taxonomic rank information for a single taxa. As default functions, these are intended for cases where the data adheres to the naming convention used by greengenes (`http://greengenes.lbl.gov/cgi-bin/nph-index.cgi`) or where the convention is unknown, respectively. To work, these functions – and any similar custom function you may want to create and use – must take as input a single character vector of taxonomic ranks for a single OTU, and return a **named** character vector that has been modified appropriately (according to known naming conventions, desired length limits, etc. The length (number of elements) of the output named vector does **not** need to be equal to the input, which is useful for the cases where the source data files have extra meaningless elements that should probably be removed, like the ubiquitous "Root" element often found in greengenes/QIIME taxonomy labels. In the case of `parse_taxonomy_default`, no naming convention is assumed and so dummy rank names are added to the vector. More usefully if your taxonomy data is based on greengenes, the `parse_taxonomy_greengenes` function clips the first 3 characters that identify the rank, and uses these to name the corresponding element according to the appropriate taxonomic rank name used by greengenes (e.g. `"p__"` at the beginning of an element means that element is the name of the phylum to which this OTU belongs). Most importantly, the expectations for these functions described above make them compatible to use during data import, specifcally the `import_biom` function, but it is a flexible structure that will be implemented soon for all phyloseq import functions that deal with taxonomy (e.g. `import_qiime`).

Usage

```
parse_taxonomy_default(char.vec)

parse_taxonomy_greengenes(char.vec)

parse_taxonomy_qiime(char.vec)
```

Arguments

`char.vec`	(Required). A single character vector of taxonomic ranks for a single OTU, unprocessed (ugly).

Value

A character vector in which each element is a different taxonomic rank of the same OTU, and each element name is the name of the rank level. For example, an element might be `"Firmicutes"` and named `"phylum"`. These parsed, named versions of the taxonomic vector should reflect embedded information, naming conventions, desired length limits, etc; or in the case of `parse_taxonomy_default`, not modified at all and given dummy rank names to each element.

See Also

`import_biom` `import_qiime`

Examples

```
taxvec1 = c("Root", "k__Bacteria", "p__Firmicutes", "c__Bacilli", "o__Bacillales", "f__Staphylococcaceae")
 parse_taxonomy_default(taxvec1)
 parse_taxonomy_greengenes(taxvec1)
taxvec2 = c("Root;k__Bacteria;p__Firmicutes;c__Bacilli;o__Bacillales;f__Staphylococcaceae")
 parse_taxonomy_qiime(taxvec2)
```

`pcoa` *S3 class for ape-calculated MDS results*

Description

Nothing to import, because ape doesn't (yet) export this S3 class. We will define it here, but keep it internal. For the moment, its only use is for proper dispatch in our extensions to the scores S3 generic from vegan, for generic extraction of coordinates and possibly other features from any ordination results.

Usage

```
pcoa
```

Format

An object of class `pcoa` of length 0.

`phylo` *S3 class placeholder definition (list) for phylogenetic trees.*

Description

The ape package does not export a version of its `phylo`-class, partly because it is not really defined formally anywhere. Instead, it is an S3 class extended from the base class, `list` – this is a very common and easy approach – and proper behavior of any method taking an instance of this class requires exact naming conventions for element names of the components. The phyloseq package does not provide any validity checks that a given phylo instance is valid (conforms to the conventions in the ape package). Yet. If problems arise, this might be considered, and they could be defined judiciously and within phyloseq. Similarly, if a formal definition for the the phylo-class is ever exported by ape, the current philosophy of phyloseq would be to remove this internal definition and import the former. Note that there is still some work going on for the phylobase package, which is addressing these same exact issues for S4 phylogenetic tree interaction. A very large number of packages (around 60 at my last count), depend on ape, making it easily the de facto standard for representing phylogenetic trees in R; and the phyloseq team would prefer to use any exported definitions from the ape package if possible and available.

Usage

```
phylo
```

Format

An object of class `phylo` of length 0.

See Also

`phylo`

`phylo-class` *An S4 placeholder of the main phylogenetic tree class from the ape package.*

Description

See the `ape` package for details about this type of representation of a phylogenetic tree. It is used throughout the ape package.

See Also

`phylo`, `setOldClass`

`phyloseq` *Build phyloseq-class objects from their components.*

Description

`phyloseq()` is a constructor method, This is the main method suggested for constructing an experiment-level (`phyloseq-class`) object from its component data (component data classes: `otu_table-class`, `sample_data-class`, `taxonomyTable-class`, `phylo-class`).

Usage

```
phyloseq(...)
```

Arguments

`...`	One or more component objects among the set of classes defined by the phyloseq package, as well as `phylo-class` (defined by the `ape-package`). Each argument should be a different class. For combining multiple components of the same class, or multiple phyloseq-class objects, use the `merge_phyloseq` function. Unlike in earlier versions, the arguments to phyloseq do not need to be named, and the order of the arguments does not matter.

Value

The class of the returned object depends on the argument class(es). For an experiment-level object, two or more component data objects must be provided. Otherwise, if a single component-class is provided, it is simply returned as-is. The order of arguments does not matter.

See Also

merge_phyloseq

Examples

```
data(esophagus)
x1 = phyloseq(otu_table(esophagus), phy_tree(esophagus))
identical(x1, esophagus)
# # data(GlobalPatterns)
# # GP <- GlobalPatterns
# # phyloseq(sample_data(GP), otu_table(GP))
# # phyloseq(otu_table(GP), phy_tree(GP))
# # phyloseq(tax_table(GP), otu_table(GP))
# # phyloseq(phy_tree(GP), otu_table(GP), sample_data(GP))
# # phyloseq(otu_table(GP), tax_table(GP), sample_data(GP))
# # phyloseq(otu_table(GP), phy_tree(GP), tax_table(GP), sample_data(GP))
```

`phyloseq-class` *The main experiment-level class for phyloseq data*

Description

Contains all currently-supported component data classes: `otu_table-class`, `sample_data-class`, `taxonomyTable-class` (`"tax_table"` slot), `phylo`-class (`"phy_tree"` slot), and the `XStringSet-class` (`"refseq"` slot). There are several advantages to storing your phylogenetic sequencing experiment as an instance of the phyloseq class, not the least of which is that it is easy to return to the data later and feel confident that the different data types “belong” to one another. Furthermore, the `phyloseq` constructor ensures that the different data components have compatible indices (e.g. OTUs and samples), and performs the necessary trimming automatically when you create your “experiment-level” object. Downstream analyses are aware of which data classes they require – and where to find them – often making your `phyloseq-class` object the only data argument required for analysis and plotting functions (although there are many options and parameter arguments available to you).

Details

In the case of missing component data, the slots are set to `NULL`. As soon as a `phyloseq-class` object is to be updated with new component data (previously missing/`NULL` or not), the indices of all components are re-checked for compatibility and trimmed if necessary. This is to ensure by design that components describe the same taxa/samples, and also that these trimming/validity checks do not need to be repeated in downstream analyses.

slots:

otu_table a single object of class otu_table.

sam_data a single object of class sample_data.

tax_table a single object of class taxonomyTable.

phy_tree a single object of the `phylo`-class, from the ape package.

refseq a biological sequence set object of a class that inherits from the `XStringSet-class`, from the Biostrings package.

See Also

The constructor, `phyloseq`, the merger `merge_phyloseq`, and also the component constructor/accessors `otu_table`, `sample_data`, `tax_table`, `phy_tree`, and `refseq`.

`phyloseq-deprecated` *Depcrecated functions in the phyloseq package.*

Description

These will be migrated to `"defunct"` status in the next release, and removed completely in the release after that. These functions are provided for compatibility with older version of the phyloseq package. They may eventually be completely removed.

Usage

```
deprecated_phyloseq_function(x, value, ...)
```

Arguments

`x`	For assignment operators, the object that will undergo a replacement (object inside parenthesis).
`value`	For assignment operators, the value to replace with (the right side of the assignment).
`...`	For functions other than assignment operators, parameters to be passed to the modern version of the function (see table).

Details

`plot_taxa_bar`	now a synonym for `plot_bar`
`taxaplot`	now a synonym for `plot_bar`
`taxtab`	now a synonym for `tax_table`
`taxTab`	now a synonym for `tax_table`
`sampleData`	now a synonym for `sample_data`
`samData`	now a synonym for `sample_data`
`sam_data`	now a synonym for `sample_data`
`speciesSums`	now a synonym for `taxa_sums`
`sampleSums`	now a synonym for `sample_sums`
`nspecies`	now a synonym for `ntaxa`
`species.names`	now a synonym for `taxa_names`
`sampleNames`	now a synonym for `sample_names`
`sample.names`	now a synonym for `sample_names`
`getSamples`	now a synonym for `get_sample`
`getSpecies`	now a synonym for `get_taxa`
`rank.names`	now a synonym for `rank_names`
`getTaxa`	now a synonym for `get_taxa_unique`
`sample.variables`	now a synonym for `sample_variables`
`getVariable`	now a synonym for `get_variable`
`merge_species`	now a synonym for `merge_taxa`
`otuTable`	now a synonym for `otu_table`

`speciesarerows`	now a synonym for `taxa_are_rows`
`speciesAreRows`	now a synonym for `taxa_are_rows`
`plot_richness_estimates`	now a synonym for `plot_richness`
`import_qiime_sampleData`	now a synonym for `import_qiime_sample_data`
`filterfunSample`	now a synonym for `filterfun_sample`
`genefilterSample`	now a synonym for `genefilter_sample`
`prune_species`	now a synonym for `prune_taxa`
`subset_species`	now a synonym for `subset_taxa`
`tipglom`	now a synonym for `tip_glom`
`taxglom`	now a synonym for `tax_glom`
`tre`	now a synonym for `phy_tree`
`show_mothur_list_cutoffs`	now a synonym for `show_mothur_cutoffs`
`sam_data<-`	now a synonym for `sample_data<-`
`sampleData<-`	now a synonym for `sample_data<-`
`tre<-`	now a synonym for `phy_tree<-`
`speciesAreRows<-`	now a synonym for `taxa_are_rows<-`
`otuTable<-`	now a synonym for `otu_table<-`
`taxTab<-`	now a synonym for `tax_table<-`

`phyloseq_to_deseq2` *Convert phyloseq data to DESeq2 dds object*

Description

No testing is performed by this function. The phyloseq data is converted to the relevant `DESeqDataSet` object, which can then be tested in the negative binomial generalized linear model framework of the `DESeq` function in DESeq2 package. See the phyloseq-extensions tutorials for more details.

Usage

```
phyloseq_to_deseq2(physeq, design, ...)
```

Arguments

`physeq`	(Required). `phyloseq-class`. Must have a `sample_data` component.
`design`	(Required). A `formula` which specifies the design of the experiment, taking the form `formula(~ x + y + z)`. That is, a formula with right-hand side only. By default, the functions in this package and DESeq2 will use the last variable in the formula (e.g. `z`) for presenting results (fold changes, etc.) and plotting. When considering your specification of experimental design, you will want to re-order the levels so that the `NULL` set is first. For example, the following line of code would ensure that Enterotype 1 is used as the reference sample class in tests by setting it to the first of the factor levels using the `relevel` function: `sample_data(entill)$Enterotype <- relevel(sample_data(entill)$Enterotype, "1")`
`...`	(Optional). Additional named arguments passed to `DESeqDataSetFromMatrix`. Most users will not need to pass any additional arguments here. Most testing-related options should be provided in a following call to `DESeq`.

Value

A `DESeqDataSet` object.

See Also

`vignette("phyloseq-mixture-models")`

The phyloseq-extensions tutorials.

`DESeq`

`results`

`DESeqDataSetFromMatrix`

Examples

```
# Check out the vignette phyloseq-mixture-models for more details.
# vignette("phyloseq-mixture-models")
data(soilrep)
phyloseq_to_deseq2(soilrep, ~warmed)
```

`phyloseq_to_metagenomeSeq`

Convert phyloseq data to MetagenomeSeq MRexperiment object

Description

No testing is performed by this function. The phyloseq data is converted to the relevant `MRexperiment-class` object, which can then be tested in the zero-inflated mixture model framework (e.g. `fitZig`) in the metagenomeSeq package. See the phyloseq-extensions tutorials for more details.

Usage

```
phyloseq_to_metagenomeSeq(physeq, ...)
```

Arguments

`physeq`	(Required). `phyloseq-class`.
`...`	(Optional). Additional named arguments passed to `newMRexperiment`. Most users will not need to pass any additional arguments here.

Value

A `MRexperiment-class` object.

See Also

`fitTimeSeries` `fitLogNormal` `fitZig` `MRtable` `MRfulltable`

Examples

```
# Check out the vignette metagenomeSeq for more details.
# vignette("metagenomeSeq")
data(soilrep)
phyloseq_to_metagenomeSeq(soilrep)
```

`phy_tree` *Retrieve phylogenetic tree (`phylo`-class) from object.*

Description

This is the suggested method for accessing the phylogenetic tree, (`phylo`-class) from a `phyloseq-class`. Like other accessors (see See Also, below), the default behavior of this method is to stop with an error if `physeq` is a `phyloseq-class` but does not contain a phylogenetic tree (the component data you are trying to access in this case).

Usage

```
phy_tree(physeq, errorIfNULL=TRUE)

## S4 method for signature 'ANY'
phy_tree(physeq, errorIfNULL = TRUE)

## S4 method for signature 'phylo'
phy_tree(physeq)
```

Arguments

`physeq`	(Required). An instance of phyloseq-class that contains a phylogenetic tree. If physeq is a phylogenetic tree (a component data class), then it is returned as-is.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Details

Note that the tip labels should be named to match the `taxa_names` of the other objects to which it is going to be paired. The `phyloseq` constructor automatically checks for exact agreement in the set of species described by the phlyogenetic tree and the other components (taxonomyTable, otu_table), and trims as-needed. Thus, the tip.labels in a phylo object must be named to match the results of `taxa_names` of the other objects to which it will ultimately be paired.

Value

The `phylo`-class object contained within `physeq`; or NULL if `physeq` does not have a tree. This method stops with an error in the latter NULL case be default, which can be over-ridden by changing the value of `errorIfNULL` to `FALSE`.

See Also

`otu_table`, `sample_data`, `tax_table` `refseq`, `phyloseq`, `merge_phyloseq`

Examples

```
data(GlobalPatterns)
phy_tree(GlobalPatterns)
```

`phy_tree<-` *Assign a (new) phylogenetic tree to* `x`

Description

Assign a (new) phylogenetic tree to `x`

Usage

```
phy_tree(x) <- value

## S4 replacement method for signature 'phyloseq,phylo'
phy_tree(x) <- value

## S4 replacement method for signature 'phyloseq,phyloseq'
phy_tree(x) <- value
```

Arguments

`x`	(Required). `phyloseq-class`
`value`	(Required). `phylo-class`, or `phyloseq-class`

Examples

```
#
data("esophagus")
# An example of pruning to just the first 20 taxa in esophagus
ex2a <- prune_taxa(taxa_names(esophagus)[1:20], esophagus)
# The following 3 lines produces an ex2b that is equal to ex2a
ex2b <- ex2a
phy_tree(ex2b) <- phy_tree(esophagus)
identical(ex2a, ex2b)
```

`plot_bar` *A flexible, informative barplot phyloseq data*

Description

There are many useful examples of phyloseq barplot graphics in the phyloseq online tutorials. This function wraps `ggplot2` plotting, and returns a `ggplot2` graphic object that can be saved or further modified with additional layers, options, etc. The main purpose of this function is to quickly and easily create informative summary graphics of the differences in taxa abundance between samples in an experiment.

Usage

```
plot_bar(physeq, x="Sample", y="Abundance", fill=NULL,
 title=NULL, facet_grid=NULL)
```

Arguments

`physeq`	(Required). An `otu_table-class` or `phyloseq-class`.
`x`	(Optional). Optional, but recommended, especially if your data is comprised of many samples. A character string. The variable in the melted-data that should be mapped to the x-axis. See `psmelt`, `melt`, and `ggplot` for more details.
`y`	(Optional). A character string. The variable in the melted-data that should be mapped to the y-axis. Typically this will be `"Abundance"`, in order to quantitatively display the abundance values for each OTU/group. However, alternative variables could be used instead, producing a very different, though possibly still informative, plot. See `psmelt`, `melt`, and `ggplot` for more details.
`fill`	(Optional). A character string. Indicates which sample variable should be used to map to the fill color of the bars. The default is `NULL`, resulting in a gray fill for all bar segments.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.
`facet_grid`	(Optional). A formula object. It should describe the faceting you want in exactly the same way as for `facet_grid`, and is ulitmately provided to `ggplot`2 graphics. The default is: `NULL`, resulting in no faceting.

Value

A `ggplot`2 graphic object – rendered in the graphical device as the default `print`/`show` method.

See Also

phyloseq online tutorials.

`psmelt`

`ggplot`

`qplot`

Examples

```
data("GlobalPatterns")
gp.ch = subset_taxa(GlobalPatterns, Phylum == "Chlamydiae")
plot_bar(gp.ch)
plot_bar(gp.ch, fill="Genus")
plot_bar(gp.ch, x="SampleType", fill="Genus")
plot_bar(gp.ch, "SampleType", fill="Genus", facet_grid=~Family)
# See additional examples in the plot_bar online tutorial. Link above.
```

`plot_clusgap` *Create a ggplot summary of gap statistic results*

Description

Create a ggplot summary of gap statistic results

Usage

```
plot_clusgap(clusgap, title = "Gap Statistic results")
```

Arguments

`clusgap`	(Required). An object of S3 class `"clusGap"`, basically a list with components. See the `clusGap` documentation for more details. In most cases this will be the output of `gapstat_ord`, or `clusGap` if you called it directly.
`title`	(Optional). Character string. The main title for the graphic. Default is `"Gap Statistic results"`.

Value

A `ggplot` plot object. The rendered graphic should be a plot of the gap statistic score versus values for k, the number of clusters.

See Also

`gapstat_ord`

`clusGap`

`ggplot`

Examples

```
# Load and process data
data("soilrep")
soilr = rarefy_even_depth(soilrep, rngseed=888)
print(soilr)
sample_variables(soilr)
# Ordination
sord  = ordinate(soilr, "DCA")
# Gap Statistic
gs = gapstat_ord(sord, axes=1:4, verbose=FALSE)
# Evaluate results with plots, etc.
plot_scree(sord)
plot_ordination(soilr, sord,  color="Treatment")
plot_clusgap(gs)
print(gs, method="Tibs2001SEmax")
# Non-ordination example, use cluster::clusGap function directly
library("cluster")
pam1 = function(x, k){list(cluster = pam(x, k, cluster.only=TRUE))}
gs.pam.RU = clusGap(ruspini, FUN = pam1, K.max = 8, B = 60)
gs.pam.RU
plot(gs.pam.RU, main = "Gap statistic for the 'ruspini' data")
mtext("k = 4 is best .. and  k = 5  pretty close")
plot_clusgap(gs.pam.RU)
```

`plot_heatmap` *Create an ecologically-organized heatmap using ggplot2 graphics*

Description

There are many useful examples of phyloseq heatmap graphics in the phyloseq online tutorials. In a 2010 article in BMC Genomics, Rajaram and Oono show describe an approach to creating a heatmap using ordination methods to organize the rows and columns instead of (hierarchical) cluster analysis. In many cases the ordination-based ordering does a much better job than h-clustering.

An immediately useful example of their approach is provided in the NeatMap package for R. The NeatMap package can be used directly on the abundance table (`otu_table-class`) of phylogenetic-sequencing data, but the NMDS or PCA ordination options that it supports are not based on ecological distances. To fill this void, phyloseq provides the `plot_heatmap()` function as an ecology-oriented variant of the NeatMap approach to organizing a heatmap and build it using ggplot2 graphics tools. The `distance` and `method` arguments are the same as for the `plot_ordination` function, and support large number of distances and ordination methods, respectively, with a strong leaning toward ecology. This function also provides the options to re-label the OTU and sample axis-ticks with a taxonomic name and/or sample variable, respectively, in the hope that this might hasten your interpretation of the patterns (See the `sample.label` and `taxa.label` documentation, below). Note that this function makes no attempt to overlay hierarchical clustering trees on the axes, as hierarchical clustering is not used to organize the plot. Also note that each re-ordered axis repeats at the edge, and so apparent clusters at the far right/left or top/bottom of the heat-map may actually be the same. For now, the placement of this edge can be considered arbitrary, so beware of this artifact of this graphical representation. If you benefit from this phyloseq-specific implementation of the NeatMap approach, please cite both our packages/articles.

Usage

```
plot_heatmap(physeq, method = "NMDS", distance = "bray",
  sample.label = NULL, taxa.label = NULL, low = "#000033",
  high = "#66CCFF", na.value = "black", trans = log_trans(4),
  max.label = 250, title = NULL, sample.order = NULL,
  taxa.order = NULL, first.sample = NULL, first.taxa = NULL, ...)
```

Arguments

`physeq`	(Required). The data, in the form of an instance of the `phyloseq-class`. This should be what you get as a result from one of the `import` functions, or any of the processing downstream. No data components beyond the `otu_table` are strictly necessary, though they may be useful if you want to re-label the axis ticks according to some observable or taxonomic rank, for instance, or if you want to use a `UniFrac`-based distance (in which case your physeq data would need to have a tree included).
`method`	(Optional). The ordination method to use for organizing the heatmap. A great deal of the usefulness of a heatmap graphic depends upon the way in which the rows and columns are ordered.
`distance`	(Optional). A character string. The ecological distance method to use in the ordination. See `distance`.
`sample.label`	(Optional). A character string. The sample variable by which you want to re-label the sample (horizontal) axis.
`taxa.label`	(Optional). A character string. The name of the taxonomic rank by which you want to re-label the taxa/species/OTU (vertical) axis. You can see available options in your data using `rank_names(physeq)`.
`low`	(Optional). A character string. An R color. See ?`colors` for options support in R (there are lots). The color that represents the lowest non-zero value in the heatmap. Default is a dark blue color, `"#000033"`.
`high`	(Optional). A character string. An R color. See `colors` for options support in R (there are lots). The color that will represent the highest value in the heatmap. The default is `"#66CCFF"`. Zero-values are treated as `NA`, and set to `"black"`, to represent a background color.

`na.value`	(Optional). A character string. An R color. See `colors` for options support in R (there are lots). The color to represent what is essentially the background of the plot, the non-observations that occur as `NA` or `0` values in the abundance table. The default is `"black"`, which works well on computer-screen graphics devices, but may be a poor choice for printers, in which case you might want this value to be `"white"`, and reverse the values of `high` and `low`, above.
`trans`	(Optional). `"trans"`-class transformer-definition object. A numerical transformer to use in the continuous color scale. See `trans_new` for details. The default is `log_trans(4)`.
`max.label`	(Optional). Integer. Default is 250. The maximum number of labeles to fit on a given axis (either x or y). If number of taxa or samples exceeds this value, the corresponding axis will be stripped of any labels. This supercedes any arguments provided to `sample.label` or `taxa.label`. Make sure to increase this value if, for example, you want a special label for an axis that has 300 indices.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.
`sample.order`	(Optional). Default `NULL`. Either a single character string matching one of the `sample_variables` in your data, or a character vector of `sample_names` in the precise order that you want them displayed in the heatmap. This overrides any ordination ordering that might be done with the `method`/`distance` arguments.
`taxa.order`	(Optional). Default `NULL`. Either a single character string matching one of the `rank_names` in your data, or a character vector of `taxa_names` in the precise order that you want them displayed in the heatmap. This overrides any ordination ordering that might be done with the `method`/`distance` arguments.
`first.sample`	(Optional). Default `NULL`. A character string matching one of the `sample_names` of your input data (`physeq`). It will become the left-most sample in the plot. For the ordination-based ordering (recommended), the left and right edges of the axes are adjaacent in a continuous ordering. Therefore, the choice of starting sample is meaningless and arbitrary, but it is aesthetically poor to have the left and right edge split a natural cluster in the data. This argument allows you to specify the left edge and thereby avoid cluster-splitting, emphasize a gradient, etc.
`first.taxa`	(Optional). Default `NULL`. A character string matching one of the `taxa_names` of your input data (`physeq`). This is equivalent to `first.sample` (above), but for the taxa/OTU indices, usually the vertical axis.
`...`	(Optional). Additional parameters passed to `ordinate`.

Details

This approach borrows heavily from the `heatmap1` function in the `NeatMap` package. Highly recommended, and we are grateful for their package and ideas, which we have adapted for our specific purposes here, but did not use an explicit dependency. At the time of the first version of this implementation, the NeatMap package depends on the rgl-package, which is not needed in phyloseq, at present. Although likely a transient issue, the rgl-package has some known installation issues that have further influenced to avoid making NeatMap a formal dependency (Although we love both NeatMap and rgl!).

Value

A heatmap plot, in the form of a `ggplot2` plot object, which can be further saved and modified.

References

Because this function relies so heavily in principle, and in code, on some of the functionality in NeatMap, please site their article if you use this function in your work.

Rajaram, S., & Oono, Y. (2010). NeatMap–non-clustering heat map alternatives in R. BMC Bioinformatics, 11, 45.

Please see further examples in the phyloseq online tutorials.

Examples

```
data("GlobalPatterns")
gpac <- subset_taxa(GlobalPatterns, Phylum=="Crenarchaeota")
# FYI, the base-R function uses a non-ecological ordering scheme,
# but does add potentially useful hclust dendrogram to the sides...
gpac <- subset_taxa(GlobalPatterns, Phylum=="Crenarchaeota")
# Remove the nearly-empty samples (e.g. 10 reads or less)
gpac = prune_samples(sample_sums(gpac) > 50, gpac)
# Arbitrary order if method set to NULL
plot_heatmap(gpac, method=NULL, sample.label="SampleType", taxa.label="Family")
# Use ordination
plot_heatmap(gpac, sample.label="SampleType", taxa.label="Family")
# Use ordination for OTUs, but not sample-order
plot_heatmap(gpac, sample.label="SampleType", taxa.label="Family", sample.order="SampleType")
# Specifying both orders omits any attempt to use ordination. The following should be the same.
p0 = plot_heatmap(gpac, sample.label="SampleType", taxa.label="Family", taxa.order="Phylum", sample.order="Sa
p1 = plot_heatmap(gpac, method=NULL, sample.label="SampleType", taxa.label="Family", taxa.order="Phylum", sam
#expect_equivalent(p0, p1)
# Example: Order matters. Random ordering of OTU indices is difficult to interpret, even with structured sample
rando = sample(taxa_names(gpac), size=ntaxa(gpac), replace=FALSE)
plot_heatmap(gpac, method=NULL, sample.label="SampleType", taxa.label="Family", taxa.order=rando, sample.ord
# # Select the edges of each axis.
# First, arbitrary edge, ordering
plot_heatmap(gpac, method=NULL)
# Second, biological-ordering (instead of default ordination-ordering), but arbitrary edge
plot_heatmap(gpac, taxa.order="Family", sample.order="SampleType")
# Third, biological ordering, selected edges
plot_heatmap(gpac, taxa.order="Family", sample.order="SampleType", first.taxa="546313", first.sample="NP2")
# Fourth, add meaningful labels
plot_heatmap(gpac, sample.label="SampleType", taxa.label="Family", taxa.order="Family", sample.order="Sample
```

`plot_net` *Microbiome Network Plot using ggplot2*

Description

There are many useful examples of phyloseq network graphics in the phyloseq online tutorials. A custom plotting function for displaying networks using advanced `ggplot`2 formatting. Note that this function is a performance and interface revision to `plot_network`, which requires an `igraph` object as its first argument. This new function is more in-line with other `plot_*` functions in the `phyloseq-package`, in that its first/main argument is a `phyloseq-class` instance. Edges in the network are created if the distance between nodes is below a (potentially arbitrary) threshold, and special care should be given to considering the choice of this threshold. However, network line thickness and opacity is scaled according to the similarity of vertices (either samples or taxa), helping to temper, somewhat, the effect of the threshold. Also note that the choice of network layout algorithm can have a large effect on the impression and interpretability of the network graphic, and you may want to familiarize yourself with some of these options (see the `laymeth` argument).

Usage

```
plot_net(physeq, distance = "bray", type = "samples", maxdist = 0.7,
  laymeth = "fruchterman.reingold", color = NULL, shape = NULL,
  rescale = FALSE, point_size = 5, point_alpha = 1,
  point_label = NULL, hjust = 1.35, title = NULL)
```

Arguments

`physeq`	(Required). The `phyloseq-class` object that you want to represent as a network.
`distance`	(Optional). Default is `"bray"`. Can be either a distance method supported by `distance`, or an already-computed `dist`-class with labels that match the indices implied by both the `physeq` and `type` arguments (that is, either sample or taxa names). If you used `distance` to pre-calculate your `dist`ance, and the same `type` argument as provided here, then they will match.
`type`	(Optional). Default `"samples"`. Whether the network represented in the primary argument, `g`, is samples or taxa/OTUs. Supported arguments are `"samples"`, `"taxa"`, where `"taxa"` indicates using the taxa indices, whether they actually represent species or some other taxonomic rank.
`maxdist`	(Optional). Default `0.7`. The maximum distance value between two vertices to connect with an edge in the graphic.
`laymeth`	(Optional). Default `"fruchterman.reingold"`. A character string that indicates the method that will determine the placement of vertices, typically based on conectedness of vertices and the number of vertices. This is an interesting topic, and there are lots of options. See `igraph-package` for related topics in general, and see `layout.auto` for descriptions of various alternative layout method options supported here. The character string argument should match exactly the layout function name with the `"layout."` omitted. Try `laymeth="list"` to see a list of options.
`color`	(Optional). Default `NULL`. The name of the sample variable in `physeq` to use for color mapping of points (graph vertices).
`shape`	(Optional). Default `NULL`. The name of the sample variable in `physeq` to use for shape mapping. of points (graph vertices).
`rescale`	(Optional). Logical. Default `FALSE`. Whether to rescale the distance values to be `[0, 1]`, in which the min value is close to zero and the max value is 1.
`point_size`	(Optional). Default `5`. The size of the vertex points.
`point_alpha`	(Optional). Default `1`. A value between 0 and 1 for the alpha transparency of the vertex points.
`point_label`	(Optional). Default `NULL`. The variable name in `physeq` covariate data to map to vertex labels.
`hjust`	(Optional). Default `1.35`. The amount of horizontal justification to use for each label.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.

Value

A ggplot2 network plot. Will render to default graphic device automatically as print side effect. Can also be saved, further manipulated, or rendered to a vector or raster file using `ggsave`.

See Also

Original network plotting functions:

`make_network`

`plot_network`

Examples

```
data(enterotype)
plot_net(enterotype, color="SeqTech", maxdist = 0.3)
plot_net(enterotype, color="SeqTech", maxdist = 0.3, laymeth = "auto")
plot_net(enterotype, color="SeqTech", maxdist = 0.3, laymeth = "svd")
plot_net(enterotype, color="SeqTech", maxdist = 0.3, laymeth = "circle")
plot_net(enterotype, color="SeqTech", shape="Enterotype", maxdist = 0.3, laymeth = "circle")
```

`plot_network` *Microbiome Network Plot using ggplot2*

Description

There are many useful examples of phyloseq network graphics in the phyloseq online tutorials. A custom plotting function for displaying networks using advanced ggplot2 formatting. The network itself should be represented using the `igraph` package. For the `phyloseq-package` it is suggested that the network object (argument g) be created using the `make_network` function, and based upon sample-wise or taxa-wise microbiome ecological distances calculated from a phylogenetic sequencing experiment (`phyloseq-class`). In this case, edges in the network are created if the distance between nodes is below a potentially arbitrary threshold, and special care should be given to considering the choice of this threshold.

Usage

```
plot_network(g, physeq=NULL, type="samples",
 color=NULL, shape=NULL, point_size=4, alpha=1,
 label="value", hjust = 1.35,
 line_weight=0.5, line_color=color, line_alpha=0.4,
 layout.method=layout.fruchterman.reingold, title=NULL)
```

Arguments

`g`	(Required). An `igraph-class` object created either by the convenience wrapper `make_network`, or directly by the tools in the igraph-package.
`physeq`	(Optional). Default `NULL`. A `phyloseq-class` object on which g is based.
`type`	(Optional). Default `"samples"`. Whether the network represented in the primary argument, g, is samples or taxa/OTUs. Supported arguments are `"samples"`, `"taxa"`, where `"taxa"` indicates using the taxa indices, whether they actually represent species or some other taxonomic rank.

`color`	(Optional). Default `NULL`. The name of the sample variable in `physeq` to use for color mapping of points (graph vertices).
`shape`	(Optional). Default `NULL`. The name of the sample variable in `physeq` to use for shape mapping. of points (graph vertices).
`point_size`	(Optional). Default `4`. The size of the vertex points.
`alpha`	(Optional). Default `1`. A value between 0 and 1 for the alpha transparency of the vertex points.
`label`	(Optional). Default `"value"`. The name of the sample variable in `physeq` to use for labelling the vertex points.
`hjust`	(Optional). Default `1.35`. The amount of horizontal justification to use for each label.
`line_weight`	(Optional). Default `0.3`. The line thickness to use to label graph edges.
`line_color`	(Optional). Default `color`. The name of the sample variable in `physeq` to use for color mapping of lines (graph edges).
`line_alpha`	(Optional). Default `0.4`. The transparency level for graph-edge lines.
`layout.method`	(Optional). Default `layout.fruchterman.reingold`. A function (closure) that determines the placement of the vertices for drawing a graph. Should be able to take an `igraph`-class as sole argument, and return a two-column coordinate matrix with `nrow` equal to the number of vertices. For possible options already included in `igraph`-package, see the others also described in the help file:
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic. `layout.fruchterman.reingold`

Value

A `ggplot`2 plot representing the network, with optional mapping of variable(s) to point color or shape.

References

This code was adapted from a repo original hosted on GitHub by Scott Chamberlain: `https://github.com/SChamberlain/gggraph`

The code most directly used/modified was first posted here: `http://www.r-bloggers.com/basic-ggplot2-network-`

See Also

`make_network`

Examples

```
data(enterotype)
ig <- make_network(enterotype, max.dist=0.3)
plot_network(ig, enterotype, color="SeqTech", shape="Enterotype", line_weight=0.3, label=NULL)
# Change distance parameter
ig <- make_network(enterotype, max.dist=0.2)
plot_network(ig, enterotype, color="SeqTech", shape="Enterotype", line_weight=0.3, label=NULL)
```

`plot_ordination` *General ordination plotter based on ggplot2.*

Description

There are many useful examples of phyloseq ordination graphics in the phyloseq online tutorials. Convenience wrapper for plotting ordination results as a `ggplot2`-graphic, including additional annotation in the form of shading, shape, and/or labels of sample variables.

Usage

```
plot_ordination(physeq, ordination, type = "samples", axes = 1:2,
  color = NULL, shape = NULL, label = NULL, title = NULL,
  justDF = FALSE)
```

Arguments

`physeq`	(Required). `phyloseq-class`. The data about which you want to plot and annotate the ordination.
`ordination`	(Required). An ordination object. Many different classes of ordination are defined by `R` packages. Ordination classes currently supported/created by the `ordinate` function are supported here. There is no default, as the expectation is that the ordination will be performed and saved prior to calling this plot function.
`type`	(Optional). The plot type. Default is `"samples"`. The currently supported options are `c("samples", "sites", "species", "taxa", "biplot", "split", "scree")`. The option "taxa" is equivalent to "species" in this case, and similarly, "samples" is equivalent to "sites". The options `"sites"` and `"species"` result in a single-plot of just the sites/samples or species/taxa of the ordination, respectively. The `"biplot"` and `"split"` options result in a combined plot with both taxa and samples, either combined into one plot ("biplot") or separated in two facet panels ("split"), respectively. The `"scree"` option results in a call to `plot_scree`, which produces an ordered bar plot of the normalized eigenvalues associated with each ordination axis.
`axes`	(Optional). A 2-element vector indicating the axes of the ordination that should be used for plotting. Can be `character-class` or `integer-class`, naming the index name or index of the desired axis for the horizontal and vertical axes, respectively, in that order. The default value, `c(1, 2)`, specifies the first two axes of the provided ordination.
`color`	(Optional). Default `NULL`. Character string. The name of the variable to map to colors in the plot. This can be a sample variable (among the set returned by `sample_variables(physeq)`) or taxonomic rank (among the set returned by `rank_names(physeq)`). Note that the color scheme is chosen automatically by `link{ggplot}`, but it can be modified afterward with an additional layer using `scale_color_manual`.
`shape`	(Optional). Default `NULL`. Character string. The name of the variable to map to different shapes on the plot. Similar to `color` option, but for the shape if points. The shape scale is chosen automatically by `link{ggplot}`, but it can be modified afterward with an additional layer using `scale_shape_manual`.

`label`	(Optional). Default `NULL`. Character string. The name of the variable to map to text labels on the plot. Similar to `color` option, but for plotting text.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.
`justDF`	(Optional). Default `FALSE`. Logical. Instead of returning a ggplot2-object, do you just want the relevant `data.frame` that was used to build the plot? This is a user-accessible option for obtaining the `data.frame`, in in principal to make a custom plot that isn't possible with the available options in this function. For contributing new functions (developers), the `phyloseq-package` provides/uses an internal function to build the key features of the `data.frame` prior to plot-build.

Value

A `ggplot` plot object, graphically summarizing the ordination result for the specified axes.

See Also

Many more examples are included in the phyloseq online tutorials.

Also see the general wrapping function:

`plot_phyloseq`

Examples

```
# See other examples at
# http://joey711.github.io/phyloseq/plot_ordination-examples
data(GlobalPatterns)
GP = prune_taxa(names(sort(taxa_sums(GlobalPatterns), TRUE)[1:50]), GlobalPatterns)
gp_bray_pcoa = ordinate(GP, "CCA", "bray")
plot_ordination(GP, gp_bray_pcoa, "samples", color="SampleType")
```

`plot_phyloseq` *Generic plot defaults for phyloseq.*

Description

There are many useful examples of phyloseq graphics functions in the phyloseq online tutorials. The specific plot type is chosen according to available non-empty slots. This is mainly for syntactic convenience and quick-plotting. See links below for some examples of available graphics tools available in the `phyloseq-package`.

Usage

```
plot_phyloseq(physeq, ...)

## S4 method for signature 'phyloseq'
plot_phyloseq(physeq, ...)
```

Arguments

`physeq`	(Required). `phyloseq-class`. The actual plot type depends on the available (non-empty) component data types contained within.
`...`	(Optional). Additional parameters to be passed on to the respective specific plotting function. See below for different plotting functions that might be called by this generic plotting wrapper.

Value

A plot is created. The nature and class of the plot depends on the physeq argument, specifically, which component data classes are present.

See Also

phyloseq frontpage tutorials.

`plot_ordination` `plot_heatmap` `plot_tree` `plot_network` `plot_bar` `plot_richness`

Examples

```
data(esophagus)
plot_phyloseq(esophagus)
```

`plot_richness` *Plot alpha diversity, flexibly with ggplot2*

Description

There are many useful examples of alpha-diversity graphics in the phyloseq online tutorials. This function estimates a number of alpha-diversity metrics using the `estimate_richness` function, and returns a `ggplot` plotting object. The plot generated by this function will include every sample in `physeq`, but they can be further grouped on the horizontal axis through the argument to `x`, and shaded according to the argument to `color` (see below). You must use untrimmed, non-normalized count data for meaningful results, as many of these estimates are highly dependent on the number of singletons. You can always trim the data later on if needed, just not before using this function.

Usage

```
plot_richness(physeq, x = "samples", color = NULL, shape = NULL,
  title = NULL, scales = "free_y", nrow = 1, shsi = NULL,
  measures = NULL, sortby = NULL)
```

Arguments

`physeq`	(Required). `phyloseq-class`, or alternatively, an `otu_table-class`. The data about which you want to estimate.
`x`	(Optional). A variable to map to the horizontal axis. The vertical axis will be mapped to the alpha diversity index/estimate and have units of total taxa, and/or index value (dimensionless). This parameter (x) can be either a character string indicating a variable in `sample_data` (among the set returned by `sample_variables(physeq)`); or a custom supplied vector with length equal to the number of samples in the dataset (nsamples(physeq)).

	The default value is `"samples"`, which will map each sample's name to a separate horizontal position in the plot.
`color`	(Optional). Default `NULL`. The sample variable to map to different colors. Like `x`, this can be a single character string of the variable name in `sample_data` (among the set returned by `sample_variables(physeq)`); or a custom supplied vector with length equal to the number of samples in the dataset (nsamples(physeq)). The color scheme is chosen automatically by `link{ggplot}`, but it can be modified afterward with an additional layer using `scale_color_manual`.
`shape`	(Optional). Default `NULL`. The sample variable to map to different shapes. Like `x` and `color`, this can be a single character string of the variable name in `sample_data` (among the set returned by `sample_variables(physeq)`); or a custom supplied vector with length equal to the number of samples in the dataset (nsamples(physeq)). The shape scale is chosen automatically by `link{ggplot}`, but it can be modified afterward with an additional layer using `scale_shape_manual`.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.
`scales`	(Optional). Default `"free_y"`. Whether to let vertical axis have free scale that adjusts to the data in each panel. This argument is passed to `facet_wrap`. If set to `"fixed"`, a single vertical scale will be used in all panels. This can obscure values if the `measures` argument includes both richness estimates and diversity indices, for example.
`nrow`	(Optional). Default is `1`, meaning that all plot panels will be placed in a single row, side-by-side. This argument is passed to `facet_wrap`. If `NULL`, the number of rows and columns will be chosen automatically (wrapped) based on the number of panels and the size of the graphics device.
`shsi`	(Deprecated). No longer supported. Instead see 'measures' below.
`measures`	(Optional). Default is `NULL`, meaning that all available alpha-diversity measures will be included in plot panels. Alternatively, you can specify one or more measures as a character vector of measure names. Values must be among those supported: `c("Observed", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson", "Fisher")`.
`sortby`	(Optional). A character string subset of `measures` argument. Sort x-indices by the mean of one or more `measures`, if x-axis is mapped to a discrete variable. Default is `NULL`, implying that a discrete-value horizontal axis will use default sorting, usually alphabetic.

Details

NOTE: Because this plotting function incorporates the output from `estimate_richness`, the variable names of that output should not be used as `x` or `color` (even if it works, the resulting plot might be kindof strange, and not the intended behavior of this function). The following are the names you will want to avoid using in `x` or `color`:

`c("Observed", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson", "Fisher")`.

Value

A `ggplot` plot object summarizing the richness estimates, and their standard error.

See Also

`estimate_richness`

`estimateR`

`diversity`

There are many more interesting examples at the phyloseq online tutorials.

Examples

```
## There are many more interesting examples at the phyloseq online tutorials.
## http://joey711.github.io/phyloseq/plot_richness-examples
data("soilrep")
plot_richness(soilrep, measures=c("InvSimpson", "Fisher"))
plot_richness(soilrep, "Treatment", "warmed", measures=c("Chao1", "ACE", "InvSimpson"), nrow=3)
data("GlobalPatterns")
plot_richness(GlobalPatterns, x="SampleType", measures=c("InvSimpson"))
plot_richness(GlobalPatterns, x="SampleType", measures=c("Chao1", "ACE", "InvSimpson"), nrow=3)
plot_richness(GlobalPatterns, x="SampleType", measures=c("Chao1", "ACE", "InvSimpson"), nrow=3, sortby = "Cha
```

`plot_scree` *General ordination eigenvalue plotter using ggplot2.*

Description

Convenience wrapper for plotting ordination eigenvalues (if available) using a `ggplot2`-graphic.

Usage

```
plot_scree(ordination, title = NULL)
```

Arguments

`ordination`	(Required). An ordination object. Many different classes of ordination are defined by `R` packages. Ordination classes currently supported/created by the `ordinate` function are supported here. There is no default, as the expectation is that the ordination will be performed and saved prior to calling this plot function.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.

Value

A `ggplot` plot object, graphically summarizing the ordination result for the specified axes.

See Also

`plot_ordination`

`ordinate`

`distance`

phyloseq online tutorials

Examples

```
# First load and trim a dataset
data("GlobalPatterns")
GP = prune_taxa(names(sort(taxa_sums(GlobalPatterns), TRUE)[1:50]), GlobalPatterns)
# Test plots (preforms ordination in-line, then makes scree plot)
plot_scree(ordinate(GP, "DPCoA", "bray"))
plot_scree(ordinate(GP, "PCoA", "bray"))
# Empty return with message
plot_scree(ordinate(GP, "NMDS", "bray"))
# Constrained ordinations
plot_scree(ordinate(GP, "CCA", formula=~SampleType))
plot_scree(ordinate(GP, "RDA", formula=~SampleType))
plot_scree(ordinate(GP, "CAP", formula=~SampleType))
# Deprecated example of constrained ordination (emits a warning)
#plot_scree(ordinate(GP ~ SampleType, "RDA"))
plot_scree(ordinate(GP, "DCA"))
plot_ordination(GP, ordinate(GP, "DCA"), type="scree")
```

`plot_tree` *Plot a phylogenetic tree with optional annotations*

Description

There are many useful examples of phyloseq tree graphics in the phyloseq online tutorials. This function is intended to facilitate easy graphical investigation of the phylogenetic tree, as well as sample data. Note that for phylogenetic sequencing of samples with large richness, some of the options in this function will be prohibitively slow to render, or too dense to be interpretable. A rough “rule of thumb” is to use subsets of data with not many more than 200 OTUs per plot, sometimes less depending on the complexity of the additional annotations being mapped to the tree. It is usually possible to create an unreadable, uninterpretable tree with modern datasets. However, the goal should be toward parameter settings and data subsets that convey (honestly, accurately) some biologically relevant feature of the data. One of the goals of the `phyloseq-package` is to make the determination of these features/settings as easy as possible.

Usage

```
plot_tree(physeq, method = "sampledodge", nodelabf = NULL,
  color = NULL, shape = NULL, size = NULL, min.abundance = Inf,
  label.tips = NULL, text.size = NULL, sizebase = 5,
  base.spacing = 0.02, ladderize = FALSE, plot.margin = 0.2,
  title = NULL, treetheme = NULL, justify = "jagged")
```

Arguments

`physeq`	(Required). The data about which you want to plot and annotate a phylogenetic tree, in the form of a single instance of the `phyloseq-class`, containing at minimum a phylogenetic tree component (try `phy_tree`). One of the major advantages of this function over basic tree-plotting utilities in the `ape`-package is the ability to easily annotate the tree with sample variables and taxonomic information. For these uses, the physeq argument should also have a `sample_data` and/or `tax_table` component(s).

`method`	(Optional). Character string. Default `"sampledodge"`. The name of the annotation method to use. This will be expanded in future versions. Currently only `"sampledodge"` and `"treeonly"` are supported. The `"sampledodge"` option results in points drawn next to leaves if individuals from that taxa were observed, and a separate point is drawn for each sample.
`nodelabf`	(Optional). A function. Default `NULL`. If `NULL`, the default, a function will be selected for you based upon whether or not there are node labels in `phy_tree(physeq)`. For convenience, the phyloseq package includes two generator functions for adding arbitrary node labels (can be any character string), `nodeplotdefault`; as well as for adding bootstrap values in a certain range, `nodeplotboot`. To not have any node labels in the graphic, set this argument to `nodeplotblank`.
`color`	(Optional). Character string. Default `NULL`. The name of the variable in `physeq` to map to point color. Supported options here also include the reserved special variables of `psmelt`.
`shape`	(Optional). Character string. Default `NULL`. The name of the variable in `physeq` to map to point shape. Supported options here also include the reserved special variables of `psmelt`.
`size`	(Optional). Character string. Default `NULL`. The name of the variable in `physeq` to map to point size. A special argument `"abundance"` is reserved here and scales point size using abundance in each sample on a log scale. Supported options here also include the reserved special variables of `psmelt`.
`min.abundance`	(Optional). Numeric. The minimum number of individuals required to label a point with the precise number. Default is `Inf`, meaning that no points will have their abundance labeled. If a vector, only the first element is used.
`label.tips`	(Optional). Character string. Default is `NULL`, indicating that no tip labels will be printed. If `"taxa_names"`, then the name of the taxa will be added to the tree; either next to the leaves, or next to the set of points that label the leaves. Alternatively, if this is one of the rank names (from `rank_names(physeq)`), then the identity (if any) for that particular taxonomic rank is printed instead.
`text.size`	(Optional). Numeric. Should be positive. The size parameter used to control the text size of taxa labels. Default is `NULL`. If left `NULL`, this function will automatically calculate a (hopefully) optimal text size given the vertical constraints posed by the tree itself. This argument is included in case the automatically-calculated size is wrong, and you want to change it. Note that this parameter is only meaningful if `label.tips` is not `NULL`.
`sizebase`	(Optional). Numeric. Should be positive. The base of the logarithm used to scale point sizes to graphically represent abundance of species in a given sample. Default is 5.
`base.spacing`	(Optional). Numeric. Default is `0.02`. Should be positive. This defines the base-spacing between points at each tip/leaf in the the tree. The larger this value, the larger the spacing between points. This is useful if you have problems with overlapping large points and/or text indicating abundance, for example. Similarly, if you don't have this problem and want tighter point-spacing, you can shrink this value.
`ladderize`	(Optional). Boolean or character string (either `FALSE`, `TRUE`, or `"left"`). Default is `FALSE`. This parameter specifies whether or not to `ladderize` the tree (i.e., reorder nodes according to the depth of their enclosed subtrees) prior to plotting. This tends to make trees more aesthetically pleasing and legible in a graphical display. When `TRUE` or `"right"`, "right" ladderization is used. When set to

	`FALSE`, no ladderization is applied. When set to `"left"`, the reverse direction (“left” ladderization) is applied. This argument is passed on to `tree_layout`.
`plot.margin`	(Optional). Numeric. Default is `0.2`. Should be positive. This defines how much right-hand padding to add to the tree plot, which can be required to not truncate tip labels. The margin value is specified as a fraction of the overall tree width which is added to the right side of the plot area. So a value of `0.2` adds twenty percent extra space to the right-hand side of the plot.
`title`	(Optional). Default `NULL`. Character string. The main title for the graphic.
`treetheme`	(Optional). A custom `ggplot2` `theme` layer to use for the tree. Supplants any default theme layers used within the function. A value of `NULL` uses a default, minimal-annotations theme. If anything other than a them or `NULL`, the current global ggplot2 theme will result.
`justify`	(Optional). A character string indicating the type of justification to use on dodged points and tip labels. A value of `"jagged"`, the default, results in these tip-mapped elements being spaced as close to the tips as possible without gaps. Currently, any other value for `justify` results in a left-justified arrangement of both labels and points.

Details

This function received an early development contribution from the work of Gregory Jordan via the ggphylo package. `plot_tree` has since been re-written. For details see `tree_layout`.

Value

A `ggplot2` plot.

See Also

`plot.phylo`

There are many useful examples of phyloseq tree graphics in the phyloseq online tutorials.

Examples

```
# # Using plot_tree() with the esophagus dataset.
# # Please note that many more interesting examples are shown
# # in the online tutorials"
# # http://joey711.github.io/phyloseq/plot_tree-examples
data(esophagus)
# plot_tree(esophagus)
# plot_tree(esophagus, color="Sample")
# plot_tree(esophagus, size="Abundance")
# plot_tree(esophagus, size="Abundance", color="samples")
plot_tree(esophagus, size="Abundance", color="Sample", base.spacing=0.03)
plot_tree(esophagus, size="abundance", color="samples", base.spacing=0.03)
```

`prune_samples` *Define a subset of samples to keep in a phyloseq object.*

Description

An S4 Generic method for pruning/filtering unwanted samples by defining those you want to keep.

Usage

```
prune_samples(samples, x)

## S4 method for signature 'character,otu_table'
prune_samples(samples, x)

## S4 method for signature 'character,sample_data'
prune_samples(samples, x)

## S4 method for signature 'character,phyloseq'
prune_samples(samples, x)

## S4 method for signature 'logical,ANY'
prune_samples(samples, x)
```

Arguments

`samples`	(Required). A character vector of the samples in object x that you want to keep – OR alternatively – a logical vector where the kept samples are TRUE, and length is equal to the number of samples in object x. If `samples` is a named logical, the samples retained is based on those names. Make sure they are compatible with the `sample_names` of the object you are modifying (`x`).
`x`	A phyloseq object.

Value

The class of the object returned by `prune_samples` matches the class of the phyloseq object, `x`.

See Also

`subset_samples`

Examples

```
 data(GlobalPatterns)
 # Subset to just the Chlamydiae phylum.
 GP.chl <- subset_taxa(GlobalPatterns, Phylum=="Chlamydiae")
 # Remove the samples that have less than 20 total reads from Chlamydiae
 GP.chl <- prune_samples(sample_sums(GP.chl)>=20, GP.chl)
# (p <- plot_tree(GP.chl, color="SampleType", shape="Family", label.tips="Genus", size="abundance"))
```

`prune_taxa` *Prune unwanted OTUs / taxa from a phylogenetic object.*

Description

An S4 Generic method for removing (pruning) unwanted OTUs/taxa from phylogenetic objects, including phylo-class trees, as well as native phyloseq package objects. This is particularly useful for pruning a phyloseq object that has more than one component that describes OTUs. Credit: the `phylo`-class version is adapted from prune.sample.

Usage

```
prune_taxa(taxa, x)

## S4 method for signature '`NULL`,ANY'
prune_taxa(taxa, x)

## S4 method for signature 'logical,ANY'
prune_taxa(taxa, x)

## S4 method for signature 'character,phylo'
prune_taxa(taxa, x)

## S4 method for signature 'character,otu_table'
prune_taxa(taxa, x)

## S4 method for signature 'character,sample_data'
prune_taxa(taxa, x)

## S4 method for signature 'character,phyloseq'
prune_taxa(taxa, x)

## S4 method for signature 'character,taxonomyTable'
prune_taxa(taxa, x)

## S4 method for signature 'character,XStringSet'
prune_taxa(taxa, x)
```

Arguments

`taxa`	(Required). A character vector of the taxa in object x that you want to keep – OR alternatively – a logical vector where the kept taxa are TRUE, and length is equal to the number of taxa in object x. If `taxa` is a named logical, the taxa retained are based on those names. Make sure they are compatible with the `taxa_names` of the object you are modifying (`x`).
`x`	(Required). A phylogenetic object, including `phylo` trees, as well as all phyloseq classes that represent taxa. If the function `taxa_names` returns a non-`NULL` value, then your object can be pruned by this function.

Value

The class of the object returned by `prune_taxa` matches the class of the argument, `x`.

See Also

`prune_samples`

prune.sample

Examples

```
data("esophagus")
esophagus
plot(sort(taxa_sums(esophagus), TRUE), type="h", ylim=c(0, 50))
x1 = prune_taxa(taxa_sums(esophagus) > 10, esophagus)
x2 = prune_taxa(names(sort(taxa_sums(esophagus), TRUE))[1:9], esophagus)
identical(x1, x2)
```

`psmelt` *Melt phyloseq data object into large data.frame*

Description

The psmelt function is a specialized melt function for melting phyloseq objects (instances of the phyloseq class), usually for producing graphics with `ggplot`2. `psmelt` relies heavily on the `melt` and `merge` functions. The naming conventions used in downstream phyloseq graphics functions have reserved the following variable names that should not be used as the names of `sample_variables` or taxonomic `rank_names`. These reserved names are `c("Sample", "Abundance", "OTU")`. Also, you should not have identical names for sample variables and taxonomic ranks. That is, the intersection of the output of the following two functions `sample_variables`, `rank_names` should be an empty vector (e.g. `intersect(sample_variables(physeq), rank_names(physeq))`). All of these potential name collisions are checked-for and renamed automtically with a warning. However, if you (re)name your variables accordingly ahead of time, it will reduce confusion and eliminate the warnings.

Usage

```
psmelt(physeq)
```

Arguments

`physeq`	(Required). An `otu_table-class` or `phyloseq-class`. Function most useful for phyloseq-class.

Details

Note that “melted” phyloseq data is stored much less efficiently, and so RAM storage issues could arise with a smaller dataset (smaller number of samples/OTUs/variables) than one might otherwise expect. For common sizes of graphics-ready datasets, however, this should not be a problem. Because the number of OTU entries has a large effect on the RAM requirement, methods to reduce the number of separate OTU entries – for instance by agglomerating OTUs based on phylogenetic distance using `tip_glom` – can help alleviate RAM usage problems. This function is made user-accessible for flexibility, but is also used extensively by plot functions in phyloseq.

Value

A `data.frame`-class table.

See Also

`plot_bar`

`melt`

`merge`

Examples

```
data("GlobalPatterns")
gp.ch = subset_taxa(GlobalPatterns, Phylum == "Chlamydiae")
mdf = psmelt(gp.ch)
nrow(mdf)
ncol(mdf)
colnames(mdf)
head(rownames(mdf))
# Create a ggplot similar to
library("ggplot2")
p = ggplot(mdf, aes(x=SampleType, y=Abundance, fill=Genus))
p = p + geom_bar(color="black", stat="identity", position="stack")
print(p)
```

`rank_names` *Retrieve the names of the taxonomic ranks*

Description

This is a simple accessor function to make it more convenient to determine the taxonomic ranks that are available in a given `phyloseq-class` object.

Usage

```
rank_names(physeq, errorIfNULL=TRUE)
```

Arguments

`physeq`	(Required). `taxonomyTable-class`, or `phyloseq-class`.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

Character vector. The names of the available taxonomic ranks.

See Also

`get_taxa` `taxa_names` `sample_names`

Examples

```
data(enterotype)
rank_names(enterotype)
```

`rarefy_even_depth` *Resample an OTU table such that all samples have the same library size.*

Description

Please note that the authors of phyloseq do not advocate using this as a normalization procedure, despite its recent popularity. Our justifications for using alternative approaches to address disparities in library sizes have been made available as an article in PLoS Computational Biology. See `phyloseq_to_deseq2` for a recommended alternative to rarefying directly supported in the phyloseq package, as well as the supplemental materials for the PLoS-CB article and the phyloseq extensions repository on GitHub. Nevertheless, for comparison and demonstration, the rarefying procedure is implemented here in good faith and with options we hope are useful. This function uses the standard R `sample` function to resample from the abundance values in the `otu_table` component of the first argument, `physeq`. Often one of the major goals of this procedure is to achieve parity in total number of counts between samples, as an alternative to other formal normalization procedures, which is why a single value for the `sample.size` is expected. This kind of resampling can be performed with and without replacement, with replacement being the more computationally-efficient, default setting. See the `replace` parameter documentation for more details. We recommended that you explicitly select a random number generator seed before invoking this function, or, alternatively, that you explicitly provide a single positive integer argument as `rngseed`.

Usage

```
rarefy_even_depth(physeq, sample.size = min(sample_sums(physeq)),
  rngseed = FALSE, replace = TRUE, trimOTUs = TRUE, verbose = TRUE)
```

Arguments

`physeq`	(Required). A `phyloseq-class` object that you want to trim/filter.
`sample.size`	(Optional). A single integer value equal to the number of reads being simulated, also known as the depth, and also equal to each value returned by `sample_sums` on the output.
`rngseed`	(Optional). A single integer value passed to `set.seed`, which is used to fix a seed for reproducibly random number generation (in this case, reproducibly random subsampling). The default value is `711`. If set to `FALSE`, then no fiddling with the RNG seed is performed, and it is up to the user to appropriately call `set.seed` beforehand to achieve reproducible results.
`replace`	(Optional). Logical. Whether to sample with replacement (`TRUE`) or without replacement (`FALSE`). The default is with replacement (`replace=TRUE`). Two implications to consider are that (1) sampling with replacement is faster and more memory efficient as currently implemented; and (2), sampling with replacement means that there is a chance that the number of reads for a given OTU in a given sample could be larger than the original count value, as opposed to sampling without replacement where the original count value is the maximum possible. Prior to phyloseq package version number `1.5.20`, this parameter did not exist and sampling with replacement was the only random subsampling implemented in the `rarefy_even_depth` function. Note that this default behavior was selected for computational efficiency, but differs from analogous functions in related packages (e.g. subsampling in QIIME).

`trimOTUs`	(Optional). `logical(1)`. Whether to trim OTUs from the dataset that are no longer observed in any sample (have a count of zero in every sample). The number of OTUs trimmed, if any, is printed to standard out as a reminder.
`verbose`	(Optional). Logical. Default is `TRUE`. If `TRUE`, extra non-warning, non-error messages are printed to standard out, describing steps in the rarefying process, the OTUs and samples removed, etc. This can be useful the first few times the function is executed, but can be set to `FALSE` as-needed once behavior has been verified as expected.

Details

This approach is sometimes mistakenly called "rarefaction", which in physics refers to a form of wave decompression; but in this context, ecology, the term refers to a repeated sampling procedure to assess species richness, first proposed in 1968 by Howard Sanders. In contrast, the procedure implemented here is used as an *ad hoc* means to normalize microbiome counts that have resulted from libraries of widely-differing sizes. Here we have intentionally adopted an alternative name, `rarefy`, that has also been used recently to describe this process and, to our knowledge, not previously used in ecology.

Make sure to use `set.seed` for exactly-reproducible results of the random subsampling.

Value

An object of class `phyloseq`. Only the `otu_table` component is modified.

See Also

`sample`

`set.seed`

Examples

```
# Test with esophagus dataset
data("esophagus")
esorepT = rarefy_even_depth(esophagus, replace=TRUE)
esorepF = rarefy_even_depth(esophagus, replace=FALSE)
sample_sums(esophagus)
sample_sums(esorepT)
sample_sums(esorepF)
## NRun Manually: Too slow!
# data("GlobalPatterns")
# GPrepT = rarefy_even_depth(GlobalPatterns, 1E5, replace=TRUE)
## Actually just this one is slow
# system.time(GPrepF <- rarefy_even_depth(GlobalPatterns, 1E5, replace=FALSE))
```

`read_tree` *Somewhat flexible tree-import function*

Description

This function is a convenience wrapper around the `read.tree` (Newick-format) and `read.nexus` (Nexus-format) importers provided by the `ape-package`. This function attempts to return a valid tree if possible using either format importer. If it fails, it silently returns `NULL` by default, rather than throwing a show-stopping error.

Usage

```
read_tree(treefile, errorIfNULL=FALSE, ...)
```

Arguments

`treefile`	(Required). A character string implying a file `connection` (like a path or URL), or an actual `connection`. Must be a Newick- or Nexus-formatted tree.
`errorIfNULL`	(Optional). Logical. Should an error be thrown if no tree can be extracted from the connection? Default is `FALSE`, indicating that `NULL` will be SILENTLY returned, rather than an error. Be cautious about this behavior. Useful for phyloseq internals, but might be hard to track in your own code if you're not aware of this "no error by default" setting. If this is a problem, change this value to `TRUE`, and you can still use the function.
`...`	(Optional). Additional parameter(s) passed to the relevant tree-importing function.

Value

If successful, returns a `phylo`-class object as defined in the `ape-package`. Returns NULL if neither tree-reading function worked.

See Also

`read_tree_greengenes`

`phylo`

`read.tree`

`read.nexus`

Examples

```
read_tree(system.file("extdata", "esophagus.tree.gz", package="phyloseq"))
read_tree(system.file("extdata", "GP_tree_rand_short.newick.gz", package="phyloseq"))
```

`read_tree_greengenes` *Read GreenGenes tree released in annotated newick format*

Description

In principal, this is a standard newick format, that can be imported into R using `read_tree`, which in-turn utilizes `read.tree`. However, `read.tree` has failed to import recent (October 2012 and later) releases of the GreenGenes tree, and this problem has been traced to the additional annotations added to some internal nodes that specify taxonomic classification between single-quotes. To solve this problem and create a clear container for fixing future problems with the format of GreenGenes-released trees, this function is available in phyloseq and exported for users. It is also referenced in the documentation of the import functions for QIIME legacy and BIOM format importers – `import_qiime` and `import_biom`, respectively. However, since the precise format of the tree is not restricted to GreenGenes trees by QIIME or for the biom-format, this function is not called automatically by those aforementioned import functions. If your tree is formatted like, or is one of, the official GreenGenes release trees, then you should use this function and provide its output to your relevant import function.

Usage

```
read_tree_greengenes(treefile)
```

Arguments

`treefile`	(Required). A character string implying a file `connection` (like a path or URL), or an actual `connection`. Must be a Newick–formatted tree released by GreenGenes in October 2012 or later. The similarity threshold of the OTUs should not matter, except that it should match your OTU table.

Value

A tree, represented as a `phylo` object.

Examples

```
# Read the May 2013, 73% similarity official tree,
# included as extra data in phyloseq.
treefile = system.file("extdata", "gg13-5-73.tree.gz", package="phyloseq")
x = read_tree_greengenes(treefile)
x
class(x)
y = read_tree(treefile)
y
class(y)
## Not run, causes an error:
# library("ape")
# read.tree(treefile)
```

`reconcile_categories` *Cleans absent levels in sample_data/data.frame.*

Description

This is used internally by the builder method, `sample_data`, to ensure that the factors describing categorical variables in a data.frame or sample_data object are free of extra levels that can plague downstream plots analysis.

Usage

```
reconcile_categories(DFSM)
```

Arguments

`DFSM`	(Required). A `data.frame` or `sample_data` object that needs to be cleaned.

Value

A single `data.frame` object. Even if the input argument is a `sample_data`, the return is a `data.frame`. Because this is intended to be used internally by the builder method, it cannot also call the builder function to re-build the cleaned `sample_data`.

Examples

```
# # # data(GlobalPatterns)
# # # SM <- sample_data(GlobalPatterns)
# # # DF <- data.frame(SM)
# # # DF <- data.frame(DF, col1=1:nrow(DF), col2=paste(1:nrow(DF), "t", sep=""))
# # # DF <- reconcile_categories(DF)
# # # SM <- sample_data(reconcile_categories(SM))
# # # sapply(DF, class)
# # # sapply(SM, class)
```

`refseq` *Retrieve reference sequences (`XStringSet`-class) from object.*

Description

This is the suggested method for accessing the phylogenetic tree, (`XStringSet`-class) from a phyloseq data object (`phyloseq-class`). Like other accessors (see See Also, below), the default behavior of this method is to stop with an error if physeq is a `phyloseq-class` but does not contain reference sequences (the component data type you are trying to access in this case).

Usage

```
refseq(physeq, errorIfNULL=TRUE)

## S4 method for signature 'ANY'
refseq(physeq, errorIfNULL = TRUE)

## S4 method for signature 'XStringSet'
refseq(physeq)
```

Arguments

`physeq`	(Required). An instance of phyloseq-class that contains a phylogenetic tree. If physeq is a phylogenetic tree (a component data class), then it is returned as-is.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

The `phylo`-class object contained within physeq; or NULL if physeq does not have a tree. This method stops with an error in the latter NULL case be default, which can be over-ridden by changing the value of `errorIfNULL` to `FALSE`.

See Also

`otu_table`, `sample_data`, `tax_table` `phy_tree`, `phyloseq`, `merge_phyloseq`

Examples

```
data(GlobalPatterns)
refseq(GlobalPatterns, FALSE)
```

`rm_outlierf` *Set to FALSE any outlier species greater than f fractional abundance.*

Description

This is for removing overly-abundant outlier taxa, not for trimming low-abundance taxa.

Usage

```
rm_outlierf(f, na.rm=TRUE)
```

Arguments

`f`	Single numeric value between 0 and 1. The maximum fractional abundance value that a taxa will be allowed to have in a sample without being marked for trimming.
`na.rm`	Logical. Should we remove NA values. Default `TRUE`.

Value

A function (enclosure), suitable for `filterfun_sample`.

See Also

`topk`, `topf`, `topp`, `rm_outlierf`

Examples

```
t1 <- 1:10; names(t1)<-paste("t", 1:10, sep="")
rm_outlierf(0.15)(t1)
## Use simulated abundance matrix
set.seed(711)
testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
taxa_sums(testOTU)
f1  <- filterfun_sample(rm_outlierf(0.1))
(wh1 <- genefilter_sample(testOTU, f1, A=1))
wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
prune_taxa(wh1, testOTU)
prune_taxa(wh2, testOTU)
```

`sample_data` *Build or access sample_data.*

Description

This is the suggested method for both constructing and accessing a table of sample-level variables (`sample_data-class`), which in the `phyloseq-package` is represented as a special extension of the `data.frame-class`. When the argument is a `data.frame`, `sample_data` will create a sample_data-class object. In this case, the rows should be named to match the `sample_names` of the other objects to which it will ultimately be paired. Alternatively, if the first argument is an experiment-level (`phyloseq-class`) object, then the corresponding `sample_data` is returned. Like other accessors (see See Also, below), the default behavior of this method is to stop with an error if `object` is a `phyloseq-class` but does not contain a `sample_data`.

Usage

```
sample_data(object, errorIfNULL=TRUE)

## S4 method for signature 'ANY'
sample_data(object, errorIfNULL = TRUE)

## S4 method for signature 'data.frame'
sample_data(object)
```

Arguments

`object`	(Required). A `data.frame-class`, or a `phyloseq-class` object.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

A `sample_data-class` object representing the sample variates of an experiment.

See Also

`phy_tree`, `tax_table`, `otu_table` `phyloseq`, `merge_phyloseq`

Examples

```
#
data(soilrep)
head(sample_data(soilrep))
```

`sample_data-class` *The S4 for storing sample variables.*

Description

Row indices represent samples, while column indices represent experimental categories, variables (and so forth) that describe the samples.

Details

.Data data-frame data, inherited from the data.frame class.

row.names Also inherited from the data.frame class; it should contain the sample names.

names Inherited from the data.frame class.

`sample_data<-` *Assign (new) sample_data to* `x`

Description

This replaces the current `sample_data` component of `x` with `value`, if `value` is a `sample_data-class`. However, if `value` is a `data.frame`, then `value` is first coerced to a `sample_data-class`, and then assigned. Alternatively, if `value` is `phyloseq-class`, then the `sample_data` component will first be accessed from `value` and then assigned. This makes possible some concise assignment/replacement statements when adjusting, modifying, or building subsets of experiment-level data. See some examples below.

Internally, this re-builds the `phyloseq-class` object using the standard `phyloseq` constructor. Thus, index mismatches between sample-describing components will not be allowed, and subsetting will occurr automatically such that only the intersection of sample IDs are included in any components. This has the added benefit of re-checking (internally) for any other issues.

Usage

```
sample_data(x) <- value
```

Arguments

`x`	(Required). `phyloseq-class`. The object to modify.
`value`	(Required). Either a `sample_data-class`, a `data.frame` that can be coerced into `sample_data-class`, or a `phyloseq-class` that contains a suitable `sample_data` component to assign to `x`. If unsure, try `sample_data(value)`, which should return a `sample_data-class` object without error.

Value

No return. This is an assignment statement.

Examples

```
data(soilrep)
soilrep
head(sample_data(soilrep))
sample_data(soilrep)$Time <- as.integer(substr(sample_data(soilrep)$Sample, 1, 1))
head(sample_data(soilrep))
```

`sample_names` *Get sample names.*

Description

Get sample names.

Usage

```
sample_names(physeq)

## S4 method for signature 'ANY'
sample_names(physeq)

## S4 method for signature 'phyloseq'
sample_names(physeq)

## S4 method for signature 'sample_data'
sample_names(physeq)

## S4 method for signature 'otu_table'
sample_names(physeq)
```

Arguments

`physeq`	(Required). A `phyloseq-class`, `sample_data`, or `otu_table-class`.

Value

A character vector. The names of the samples in physeq.

See Also

`taxa_names`, `nsamples`

Examples

```
data(esophagus)
sample_names(esophagus)
```

`sample_names<-` *Replace OTU identifier names*

Description

Replace OTU identifier names

Usage

```
sample_names(x) <- value

## S4 replacement method for signature 'ANY,ANY'
sample_names(x) <- value

## S4 replacement method for signature 'ANY,character'
sample_names(x) <- value

## S4 replacement method for signature 'otu_table,character'
sample_names(x) <- value
```

```
## S4 replacement method for signature 'sample_data,character'
sample_names(x) <- value

## S4 replacement method for signature 'phyloseq,character'
sample_names(x) <- value
```

Arguments

x	(Required). An object defined by the `phyloseq-package` that describes OTUs in some way.
value	(Required). A character vector to replace the current `sample_names`.

Examples

```
data("esophagus")
sample_names(esophagus)
# plot_tree(esophagus, color="sample_names", ladderize="left")
sample_names(esophagus) <- paste("Sa-", sample_names(esophagus), sep="")
sample_names(esophagus)
# plot_tree(esophagus, color="sample_names", ladderize="left")
## non-characters are first coerced to characters.
sample_names(esophagus) <- 1:nsamples(esophagus)
sample_names(esophagus)
# plot_tree(esophagus, color="sample_names", ladderize="left")
## Cannot assign non-unique or differently-lengthed name vectors. Error.
# sample_names(esophagus) <- sample(c(TRUE, FALSE), nsamples(esophagus), TRUE)
# sample_names(esophagus) <- sample(sample_names(esophagus), nsamples(esophagus)-1, FALSE)
```

`sample_sums` *Returns the total number of individuals observed from each sample.*

Description

A convenience function equivalent to rowSums or colSums, but where the orientation of the otu_table is automatically handled.

Usage

```
sample_sums(x)
```

Arguments

x	`otu_table-class`, or `phyloseq-class`.

Value

A named `numeric-class` length equal to the number of samples in the `x`, name indicating the sample ID, and value equal to the sum of all individuals observed for each sample in `x`.

See Also

`taxa_sums`, `rowSums`, `colSums`

Examples

```
data(enterotype)
sample_sums(enterotype)
data(esophagus)
sample_sums(esophagus)
```

`sample_variables` *Get the sample variables present in sample_data*

Description

This is a simple accessor function to make it more convenient to determine the sample variable names of a particular `phyloseq-class` object.

Usage

```
sample_variables(physeq, errorIfNULL=TRUE)
```

Arguments

`physeq`	(Required). `sample_data-class`, or `phyloseq-class`.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

Character vector. The names of the variables in the sample_data data.frame. Essentially the column names. Useful for selecting model and graphics parameters that interact with sample_data.

See Also

`get_taxa` `taxa_names` `sample_names`

Examples

```
data(enterotype)
sample_variables(enterotype)
```

`show,otu_table-method` *method extensions to show for phyloseq objects.*

Description

See the general documentation of `show` method for expected behavior.

Usage

```
## S4 method for signature 'otu_table'
show(object)

## S4 method for signature 'sample_data'
show(object)

## S4 method for signature 'taxonomyTable'
show(object)

## S4 method for signature 'phyloseq'
show(object)
```

Arguments

`object`	Any R object

See Also

`show`

Examples

```
# data(GlobalPatterns)
# show(GlobalPatterns)
# GlobalPatterns
```

`show_mothur_cutoffs` *Show cutoff values available in a mothur file.*

Description

This is a helper function to report back to the user the different cutoff values available in a given mothur file – for instance, a list or shared file.

Usage

```
show_mothur_cutoffs(mothur_list_file)
```

Arguments

`mothur_list_file`	The file name and/or location as produced by *mothur*.

Value

A character vector of the different cutoff values contained in the file. For a given set of arguments to the `cluster()` command from within *mothur*, a number of OTU-clustering results are returned in the same file. The exact cutoff values used by *mothur* can vary depending on the input data/parameters. This simple function returns the cutoffs that were actually included in the *mothur* output. This an important extra step prior to importing data with the `import_mothur` function.

See Also

`import_mothur`

---

splat.phyloseq.objects

Convert `phyloseq-class` *into a named list of its non-empty components.*

Description

This is used in internal handling functions, and one of its key features is that the names in the returned-list match the slot-names, which is useful for constructing calls with language-computing functions like `do.call`. Another useful aspect is that it only returns the contents of non-empty slots. In general, this should only be used by phyloseq-package developers. Standard users should not need or use this function, and should use the accessors and other tools that leave the multi-component object in one piece.

Usage

```
splat.phyloseq.objects(x)
```

Arguments

`x`	A `phyloseq-class` object. Alternatively, a component data object will work, resulting in named list of length 1.

Value

A named list, where each element is a component object that was contained in the argument, `x`. Each element is named according to its slot-name in the phyloseq-object from which it is derived. If `x` is already a component data object, then a list of length (1) is returned, also named.

See Also

merge_phyloseq

Examples

```
#
```

`subset_ord_plot` *Subset points from an ordination-derived ggplot*

Description

Easily retrieve a plot-derived `data.frame` with a subset of points according to a threshold and method. The meaning of the threshold depends upon the method. See argument description below. There are many useful examples of phyloseq ordination graphics in the phyloseq online tutorials.

Usage

```
subset_ord_plot(p, threshold=0.05, method="farthest")
```

Arguments

`p`	(Required). A `ggplot` object created by `plot_ordination`. It contains the complete data that you want to subset.
`threshold`	(Optional). A numeric scalar. Default is `0.05`. This value determines a coordinate threshold or population threshold, depending on the value of the `method` argument, ultimately determining which points are included in returned `data.frame`.
`method`	(Optional). A character string. One of `c("farthest", "radial", "square")`. Default is `"farthest"`. This determines how threshold will be interpreted. **farthest** Unlike the other two options, this option implies removing a certain fraction or number of points from the plot, depending on the value of `threshold`. If `threshold` is greater than or equal to `1`, then all but `threshold` number of points farthest from the origin are removed. Otherwise, if `threshold` is less than `1`, all but `threshold` fraction of points farthests from origin are retained. **radial** Keep only those points that are beyond `threshold` radial distance from the origin. Has the effect of removing a circle of points from the plot, centered at the origin. **square** Keep only those points with at least one coordinate greater than `threshold`. Has the effect of removing a "square" of points from the plot, centered at the origin.

Value

A `data.frame` suitable for creating a `ggplot` plot object, graphically summarizing the ordination result according to previously-specified parameters.

See Also

phyloseq online tutorial for this function.

`plot_ordination`

Examples

```
## See the online tutorials.
## http://joey711.github.io/phyloseq/subset_ord_plot-examples
```

`subset_samples` *Subset samples by sample_data expression*

Description

This is a convenience wrapper around the `subset` function. It is intended to allow subsetting complex experimental objects with one function call. Subsetting is based on an expression for which the context first includes the variables contained in `sample_data`. The `samples` retained in the dataset is equivalent to `x[subset & !is.na(subset)]`, where `x` is the vector of sample IDs and `subset` is the logical that results from your subsetting expression. This is important to keep in mind, as users are often unaware that this subsetting step also removes/omits samples that have a missing value, `NA`, somewhere in the expression.

Usage

```
subset_samples(physeq, ...)
```

Arguments

`physeq`	A `sample_data-class`, or a `phyloseq-class` object with a `sample_data`. If the `sample_data` slot is missing in `physeq`, then `physeq` will be returned as-is, and a warning will be printed to screen.
`...`	The subsetting expression that should be applied to the `sample_data`. This is passed on to `subset`, see its documentation for more details.

Value

A subsetted object with the same class as `physeq`.

See Also

`subset_species`

Examples

```
# data(GlobalPatterns)
# subset_samples(GlobalPatterns, SampleType=="Ocean")
```

`subset_taxa` *Subset species by taxonomic expression*

Description

This is a convenience wrapper around the `subset` function. It is intended to speed subsetting complex experimental objects with one function call. In the case of `subset_taxa`, the subsetting will be based on an expression related to the columns and values within the `tax_table` (`taxonomyTable` component) slot of `physeq`. The `OTUs` retained in the dataset is equivalent to `x[subset & !is.na(subset)]`, where `x` is the vector of OTU IDs and `subset` is the logical that results from your subsetting expression. This is important to keep in mind, as users are often unaware that this subsetting step also removes/omits OTUs that have a missing value result, `NA`, somewhere in the expression.

Usage

```
subset_taxa(physeq, ...)
```

Arguments

`physeq`	A `taxonomyTable-class`, or `phyloseq-class` that contains a taxonomyTable. If the `tax_table` slot is missing in `physeq`, then `physeq` will be returned as-is and a warning will be printed to screen.
`...`	The subsetting expression that should be applied to the `taxonomyTable`. This is passed on to `subset`, and more details and examples about how it functions can be found in its documentation.

Value

A subsetted object with the same class as `physeq`.

See Also

`subset_samples`

Examples

```
## ex3 <- subset_taxa(GlobalPatterns, Phylum=="Bacteroidetes")
```

`t` *Transpose* `otu_table-class` *or* `phyloseq-class`

Description

Extends the base transpose method, `t`.

Usage

```
t(x)

## S4 method for signature 'otu_table'
t(x)

## S4 method for signature 'phyloseq'
t(x)
```

Arguments

`x`	An `otu_table` or `phyloseq-class`.

Value

The class of the object returned by `t` matches the class of the argument, `x`. The `otu_table` is transposed, and `taxa_are_rows` value is toggled.

Examples

```
data(GlobalPatterns)
otu_table(GlobalPatterns)
t( otu_table(GlobalPatterns) )
```

`taxa_are_rows` *Access taxa_are_rows slot from otu_table objects.*

Description

Access taxa_are_rows slot from otu_table objects.

Usage

```
taxa_are_rows(physeq)

## S4 method for signature 'ANY'
taxa_are_rows(physeq)

## S4 method for signature 'otu_table'
taxa_are_rows(physeq)

## S4 method for signature 'phyloseq'
taxa_are_rows(physeq)
```

Arguments

`physeq`	(Required). `phyloseq-class`, or `otu_table-class`.

Value

A logical indicating the orientation of the otu_table.

See Also

`otu_table`

`taxa_are_rows<-` *Manually change taxa_are_rows through assignment.*

Description

The taxa_are_rows slot is a logical indicating the orientation of the abundance table contained in object `x`.

Usage

```
taxa_are_rows(x) <- value

## S4 replacement method for signature 'otu_table,logical'
taxa_are_rows(x) <- value

## S4 replacement method for signature 'phyloseq,logical'
taxa_are_rows(x) <- value
```

Arguments

`x`	`otu_table-class` or `phyloseq-class`
`value`	A logical of length equal to 1. If `length(value)` > 1, the additional elements will be ignored. Only the first element is assigned to the taxa_are_rows slot.

Examples

```
data(esophagus)
taxa_are_rows(esophagus)
taxa_are_rows(otu_table(esophagus))
```

`taxa_names` *Get species / taxa names.*

Description

Get species / taxa names.

Usage

```
taxa_names(physeq)

## S4 method for signature 'ANY'
taxa_names(physeq)

## S4 method for signature 'phyloseq'
taxa_names(physeq)

## S4 method for signature 'otu_table'
taxa_names(physeq)

## S4 method for signature 'taxonomyTable'
taxa_names(physeq)

## S4 method for signature 'sample_data'
taxa_names(physeq)

## S4 method for signature 'phylo'
taxa_names(physeq)

## S4 method for signature 'XStringSet'
taxa_names(physeq)
```

Arguments

physeq	phyloseq-class, otu_table-class, taxonomyTable-class, or phylo

Value

A character vector of the names of the species in physeq.

See Also

ntaxa

Examples

```
#
data("esophagus")
tree <- phy_tree(esophagus)
OTU1 <- otu_table(esophagus)
taxa_names(tree)
taxa_names(OTU1)
physeq1 <- phyloseq(OTU1, tree)
taxa_names(physeq1)
```

`taxa_names<-` *Replace OTU identifier names*

Description

Replace OTU identifier names

Usage

```
taxa_names(x) <- value

## S4 replacement method for signature 'ANY,ANY'
taxa_names(x) <- value

## S4 replacement method for signature 'ANY,character'
taxa_names(x) <- value

## S4 replacement method for signature 'otu_table,character'
taxa_names(x) <- value

## S4 replacement method for signature 'taxonomyTable,character'
taxa_names(x) <- value

## S4 replacement method for signature 'phylo,character'
taxa_names(x) <- value

## S4 replacement method for signature 'XStringSet,character'
taxa_names(x) <- value

## S4 replacement method for signature 'phyloseq,character'
taxa_names(x) <- value
```

Arguments

`x`	(Required). An object defined by the `phyloseq-package` that describes OTUs in some way.
`value`	(Required). A character vector to replace the current `taxa_names`.

Examples

```
data("esophagus")
taxa_names(esophagus)
# plot_tree(esophagus, label.tips="taxa_names", ladderize="left")
taxa_names(esophagus) <- paste("OTU-", taxa_names(esophagus), sep="")
taxa_names(esophagus)
# plot_tree(esophagus, label.tips="taxa_names", ladderize="left")
## non-characters are first coerced to characters.
taxa_names(esophagus) <- 1:ntaxa(esophagus)
taxa_names(esophagus)
# plot_tree(esophagus, label.tips="taxa_names", ladderize="left")
## Cannot assign non-unique or differently-lengthed name vectors. Error.
# taxa_names(esophagus) <- sample(c(TRUE, FALSE), ntaxa(esophagus), TRUE)
# taxa_names(esophagus) <- sample(taxa_names(esophagus), ntaxa(esophagus)-5, FALSE)
```

`taxa_sums` *Returns the total number of individuals observed from each species/taxa/OTU.*

Description

A convenience function equivalent to rowSums or colSums, but where the orientation of the otu_table is automatically handled.

Usage

```
taxa_sums(x)
```

Arguments

`x`	`otu_table-class`, or `phyloseq-class`.

Value

A `numeric-class` with length equal to the number of species in the table, name indicated the taxa ID, and value equal to the sum of all individuals observed for each taxa in `x`.

See Also

`sample_sums`, `rowSums`, `colSums`

Examples

```
data(enterotype)
taxa_sums(enterotype)
data(esophagus)
taxa_sums(esophagus)
```

`taxonomyTable-class` *An S4 class that holds taxonomic classification data as a character matrix.*

Description

Row indices represent taxa, columns represent taxonomic classifiers.

Details

.Data This slot is inherited from the `matrix` class.

`tax_glom` *Agglomerate taxa of the same type.*

Description

This method merges species that have the same taxonomy at a certain taxonomic rank. Its approach is analogous to `tip_glom`, but uses categorical data instead of a tree. In principal, other categorical data known for all taxa could also be used in place of taxonomy, but for the moment, this must be stored in the `taxonomyTable` of the data. Also, columns/ranks to the right of the rank chosen to use for agglomeration will be replaced with `NA`, because they should be meaningless following agglomeration.

Usage

```
tax_glom(physeq, taxrank=rank_names(physeq)[1], NArm=TRUE, bad_empty=c(NA, "", " ", "\t"))
```

Arguments

`physeq`	(Required). `phyloseq-class` or `otu_table`.
`taxrank`	A character string specifying the taxonomic level that you want to agglomerate over. Should be among the results of `rank_names(physeq)`. The default value is `rank_names(physeq)[1]`, which may agglomerate too broadly for a given experiment. You are strongly encouraged to try different values for this argument.
`NArm`	(Optional). Logical, length equal to one. Default is `TRUE`. CAUTION. The decision to prune (or not) taxa for which you lack categorical data could have a large effect on downstream analysis. You may want to re-compute your analysis under both conditions, or at least think carefully about what the effect might be and the reasons explaining the absence of information for certain taxa. In the case of taxonomy, it is often a result of imprecision in taxonomic designation based on short phylogenetic sequences and a patchy system of nomenclature. If this seems to be an issue for your analysis, think about also trying the nomenclature-agnostic `tip_glom` method if you have a phylogenetic tree available.
`bad_empty`	(Optional). Character vector. Default: `c(NA, "", " ", "\t")`. Defines the bad/empty values that should be ignored and/or considered unknown. They will be removed from the internal agglomeration vector derived from the argument to `tax`, and therefore agglomeration will not combine taxa according to the presence of these values in `tax`. Furthermore, the corresponding taxa can be optionally pruned from the output if `NArm` is set to `TRUE`.

Value

A taxonomically-agglomerated, optionally-pruned, object with class matching the class of physeq.

See Also

`tip_glom`

`prune_taxa`

`merge_taxa`

Examples

```
# data(GlobalPatterns)
# ## print the available taxonomic ranks
# colnames(tax_table(GlobalPatterns))
# ## agglomerate at the Family taxonomic rank
# (x1 <- tax_glom(GlobalPatterns, taxrank="Family") )
# ## How many taxa before/after agglomeration?
# ntaxa(GlobalPatterns); ntaxa(x1)
# ## Look at enterotype dataset...
# data(enterotype)
# ## print the available taxonomic ranks. Shows only 1 rank available, not useful for tax_glom
# colnames(tax_table(enterotype))
```

`tax_table` *Build or access the taxonomyTable.*

Description

This is the suggested method for both constructing and accessing a table of taxonomic names, organized with ranks as columns (`taxonomyTable-class`). When the argument is a character matrix, tax_table() will create and return a `taxonomyTable-class` object. In this case, the rows should be named to match the `species.names` of the other objects to which it will ultimately be paired. Alternatively, if the first argument is an experiment-level (`phyloseq-class`) object, then the corresponding `taxonomyTable` is returned. Like other accessors (see See Also, below), the default behavior of this method is to stop with an error if `object` is a `phyloseq-class` but does not contain a `taxonomyTable`.

Usage

```
tax_table(object, errorIfNULL=TRUE)

## S4 method for signature 'ANY'
tax_table(object, errorIfNULL = TRUE)

## S4 method for signature 'matrix'
tax_table(object)

## S4 method for signature 'data.frame'
tax_table(object)
```

Arguments

`object`	An object among the set of classes defined by the phyloseq package that contain taxonomyTable.
`errorIfNULL`	(Optional). Logical. Should the accessor stop with an error if the slot is empty (`NULL`)? Default `TRUE`.

Value

A `taxonomyTable-class` object. It is either grabbed from the relevant slot if `object` is complex, or built anew if `object` is a character matrix representing the taxonomic classification of species in the experiment.

See Also

`phy_tree`, `sample_data`, `otu_table` `phyloseq`, `merge_phyloseq`

Examples

```
#
# tax1 <- tax_table(matrix("abc", 30, 8))
# data(GlobalPatterns)
# tax_table(GlobalPatterns)
```

`tax_table<-` *Assign a (new) Taxonomy Table to* `x`

Description

Assign a (new) Taxonomy Table to `x`

Usage

```
tax_table(x) <- value

## S4 replacement method for signature 'phyloseq,taxonomyTable'
tax_table(x) <- value

## S4 replacement method for signature 'phyloseq,ANY'
tax_table(x) <- value

## S4 replacement method for signature 'taxonomyTable,taxonomyTable'
tax_table(x) <- value

## S4 replacement method for signature 'taxonomyTable,ANY'
tax_table(x) <- value
```

Arguments

`x`	(Required). `phyloseq-class`
`value`	(Required). `taxonomyTable-class`. Alternatively, `value` can be a `phyloseq-class` that has a `tax_table` component, or a `matrix-class` that can be coerced to a `taxonomyTable-class` with row indices that match at least some of the `taxa_names` of `x`.

Examples

```
# data(GlobalPatterns)
# # An example of pruning to just the first 100 taxa in GlobalPatterns.
# ex2a <- prune_taxa(taxa_names(GlobalPatterns)[1:100], GlobalPatterns)
# # The following 3 lines produces an ex2b that is equal to ex2a
# ex2b <- GlobalPatterns
# TT <- tax_table(GlobalPatterns)[1:100, ]
# tax_table(ex2b) <- TT
# identical(ex2a, ex2b)
# print(ex2b)
# # 2 examples adding a tax_table component from phyloseq or matrix classes
# ex2c <- phyloseq(otu_table(ex2b), sample_data(ex2b), phy_tree(ex2b))
# tax_table(ex2c) <- ex2b
# identical(ex2a, ex2c)
# ex2c <- phyloseq(otu_table(ex2b), sample_data(ex2b), phy_tree(ex2b))
# tax_table(ex2c) <- as(tax_table(ex2b), "matrix")
# identical(ex2a, ex2c)
```

`threshrank` *Thresholded rank transformation.*

Description

The lowest `thresh` values in `x` all get the value 'thresh'.

Usage

```
threshrank(x, thresh, keep0s=FALSE, ...)
```

Arguments

`x`	(Required). Numeric vector to transform.
`thresh`	A single numeric value giving the threshold.
`keep0s`	A logical determining whether 0's in x should remain a zero-value in the output. If FALSE, zeros are treated as any other value.
`...`	Further arguments passes to the `rank` function.

Value

A ranked, (optionally) thresholded numeric vector with length equal to `x`. Default arguments to `rank` are used, unless provided as additional arguments.

See Also

`transform_sample_counts`, `rank`, `threshrankfun`

Examples

```
#
(a_vector <- sample(0:10, 100, TRUE))
threshrank(a_vector, 5, keep0s=TRUE)
data(GlobalPatterns)
GP <- GlobalPatterns
## These three approaches result in identical otu_table
(x1 <- transform_sample_counts( otu_table(GP), threshrankfun(500)) )
(x2 <- otu_table(apply(otu_table(GP), 2, threshrankfun(500)), taxa_are_rows(GP)) )
identical(x1, x2)
(x3 <- otu_table(apply(otu_table(GP), 2, threshrank, thresh=500), taxa_are_rows(GP)) )
identical(x1, x3)
```

`threshrankfun` *A closure version of the* `threshrank` *function.*

Description

Takes the same arguments as `threshrank`, except for `x`, because the output is a single-argument function rather than a rank-transformed numeric. This is useful for higher-order functions that require a single-argument function as input, like `transform_sample_counts`.

Usage

```
threshrankfun(thresh, keep0s=FALSE, ...)
```

Arguments

`thresh`	A single numeric value giving the threshold.
`keep0s`	A logical determining whether 0's in `x` should remain a zero-value in the output. If FALSE, zeros are treated as any other value.
`...`	Further arguments passes to the `rank` function.

Value

A single-argument function with the options to `threshrank` set.

See Also

`transform_sample_counts`, `threshrankfun`, `threshrank`

Examples

```
data(esophagus)
x1 = transform_sample_counts(esophagus, threshrankfun(50))
otu_table(x1)
x2 = transform_sample_counts(esophagus, rank)
otu_table(x2)
identical(x1, x2)
```

`tip_glom` *Agglomerate closely-related taxa using single-linkage clustering.*

Description

All tips of the tree separated by a cophenetic distance smaller than `h` will be agglomerated into one taxa using `merge_taxa`.

Usage

```
tip_glom(physeq, h = 0.2, hcfun = agnes, ...)
```

Arguments

`physeq`	(Required). A `phyloseq-class`, containing a phylogenetic tree. Alternatively, a phylogenetic tree `phylo` will also work.
`h`	(Optional). Numeric scalar of the height where the tree should be cut. This refers to the tree resulting from hierarchical clustering of `cophenetic.phylo(phy_tree(physeq))`, not necessarily the original phylogenetic tree, `phy_tree(physeq)`. Default value is `0.2`. Note that this argument used to be named `speciationMinLength`, before this function/method was rewritten.
`hcfun`	(Optional). A function. The (agglomerative, hierarchical) clustering function to use. Good examples are `agnes` and `hclust`. The default is `agnes`.
`...`	(Optional). Additional named arguments to pass to `hcfun`.

Details

Can be used to create a non-trivial OTU Table, if a phylogenetic tree is available.

For now, a simple, "greedy", single-linkage clustering is used. In future releases it should be possible to specify different clustering approaches available in `R`, in particular, complete-linkage clustering appears to be used more commonly for OTU clustering applications.

Value

An instance of the `phyloseq-class`. Or alternatively, a `phylo` object if the `physeq` argument was just a tree. In the expected-use case, the number of OTUs will be fewer (see `ntaxa`), after merging OTUs that are related enough to be called the same OTU.

See Also

`merge_taxa`

`agnes`

`hclust`

`cophenetic.phylo`

`phylo`

Examples

```
data("esophagus")
# for speed
esophagus = prune_taxa(taxa_names(esophagus)[1:25], esophagus)
plot_tree(esophagus, label.tips="taxa_names", size="abundance", title="Before tip_glom()")
plot_tree(tip_glom(esophagus, h=0.2), label.tips="taxa_names", size="abundance", title="After tip_glom()")
```

`topf` *Make filter fun. that returns the top f fraction of taxa in a sample.*

Description

As opposed to topp, which gives the most abundant p fraction of observed taxa (richness, instead of cumulative abundance. Said another way, topf ensures a certain fraction of the total sequences are retained, while topp ensures that a certain fraction of taxa/species/OTUs are retained.

Usage

```
topf(f, na.rm=TRUE)
```

Arguments

`f`	Single numeric value between 0 and 1.
`na.rm`	Logical. Should we remove NA values. Default `TRUE`.

Value

A function (enclosure), suitable for `filterfun_sample`, that will return `TRUE` for each element in the taxa comprising the most abundant f fraction of individuals.

See Also

`topk`, `topf`, `topp`, `rm_outlierf`

Examples

```
t1 <- 1:10; names(t1)<-paste("t", 1:10, sep="")
topf(0.6)(t1)
## Use simulated abundance matrix
set.seed(711)
testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
f1  <- filterfun_sample(topf(0.4))
(wh1 <- genefilter_sample(testOTU, f1, A=1))
wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
prune_taxa(wh1, testOTU)
prune_taxa(wh2, testOTU)
```

`topk` *Make filter fun. the most abundant* `k` *taxa*

Description

Make filter fun. the most abundant `k` taxa

Usage

```
topk(k, na.rm=TRUE)
```

Arguments

`k`	An integer, indicating how many of the most abundant taxa should be kept.
`na.rm`	A logical. Should `NA`s be removed. Default is `TRUE`.

Value

Returns a function (enclosure) that will return TRUE for each element in the most abundant k values.

See Also

`topk`, `topf`, `topp`, `rm_outlierf`

Examples

```
## Use simulated abundance matrix
set.seed(711)
testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
f1  <- filterfun_sample(topk(2))
wh1 <- genefilter_sample(testOTU, f1, A=2)
wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
prune_taxa(wh1, testOTU)
prune_taxa(wh2, testOTU)
```

`topp` *Make filter fun. that returns the most abundant* `p` *fraction of taxa*

Description

Make filter fun. that returns the most abundant `p` fraction of taxa

Usage

```
topp(p, na.rm=TRUE)
```

Arguments

`p`	A numeric of length 1, indicating what fraction of the most abundant taxa should be kept.
`na.rm`	A logical. Should `NA`s be removed. Default is `TRUE`.

Value

A function (enclosure), suitable for `filterfun_sample`, that will return `TRUE` for each element in the most abundant p fraction of taxa.

See Also

`topk`, `topf`, `topp`, `rm_outlierf`

Examples

```
## Use simulated abundance matrix
set.seed(711)
testOTU <- otu_table(matrix(sample(1:50, 25, replace=TRUE), 5, 5), taxa_are_rows=FALSE)
sample_sums(testOTU)
f1  <- filterfun_sample(topp(0.2))
(wh1 <- genefilter_sample(testOTU, f1, A=1))
wh2 <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
prune_taxa(wh1, testOTU)
prune_taxa(wh2, testOTU)
```

`transform_sample_counts`
Transform abundance data in an `otu_table`*, sample-by-sample.*

Description

This function transforms the sample counts of a taxa abundance matrix according to a user-provided function. The counts of each sample will be transformed individually. No sample-sample interaction/comparison is possible by this method.

Usage

```
transform_sample_counts(physeq, fun, ...)

transformSampleCounts(physeq, fun, ...)
```

Arguments

`physeq`	(Required). `phyloseq-class` of `otu_table-class`.
`fun`	(Required). A single-argument function that will be applied to the abundance counts of each sample. Can be an anonymous `function`.
`...`	(Optional). Additional, optionally-named, arguments passed to `fun` during transformation of abundance data.

Value

A transformed `otu_table` – or `phyloseq` object with its transformed `otu_table`. In general, trimming is not expected by this method, so it is suggested that the user provide only functions that return a full-length vector. Filtering/trimming can follow, for which the `genefilter_sample` and `prune_taxa` functions are suggested.

See Also

threshrankfun, rank, log

Examples

```
#
data(esophagus)
x1 = transform_sample_counts(esophagus, threshrankfun(50))
head(otu_table(x1), 10)
x2 = transform_sample_counts(esophagus, rank)
head(otu_table(x2), 10)
identical(x1, x2)
x3 = otu_table(esophagus) + 5
x3 = transform_sample_counts(x3, log)
head(otu_table(x3), 10)
x4 = transform_sample_counts(esophagus, function(x) round(x^2.2, 0))
head(otu_table(x4), 10)
```

`tree_layout` *Returns a data table defining the line segments of a phylogenetic tree.*

Description

This function takes a `phylo` or `phyloseq-class` object and returns a list of two `data.tables` suitable for plotting a phylogenetic tree with `ggplot2`.

Usage

```
tree_layout(phy, ladderize = FALSE)
```

Arguments

`phy`	(Required). The `phylo` or `phyloseq-class` object (which must contain a `phylo`genetic tree) that you want to converted to `data.tables` suitable for plotting with `ggplot2`.
`ladderize`	(Optional). Boolean or character string (either `FALSE`, `TRUE`, or `"left"`). Default is `FALSE` (no ladderization). This parameter specifies whether or not to `ladderize` the tree (i.e., reorder nodes according to the depth of their enclosed subtrees) prior to plotting. This tends to make trees more aesthetically pleasing and legible in a graphical display. When `TRUE` or `"right"`, "right" ladderization is used. When set to `FALSE`, no ladderization is applied. When set to `"left"`, the reverse direction ("left" ladderization) is applied.

Value

A list of two `data.tables`, containing respectively a `data.table` of edge segment coordinates, named `edgeDT`, and a `data.table` of vertical connecting segments, named `vertDT`. See `example` below for a simple demonstration.

See Also

An early example of this functionality was borrowed directly, with permission, from the package called ggphylo, released on GitHub at: https://github.com/gjuggler/ggphylo by its author Gregory Jordan <gjuggler@gmail.com>. That original phyloseq internal function, `tree.layout`, has been completely replaced by this smaller and much faster user-accessible function that utilizes performance enhancements from standard `data.table` magic as well as `ape-package` internal C code.

Examples

```
library("ggplot2")
data("esophagus")
phy = phy_tree(esophagus)
phy <- ape::root(phy, "65_2_5", resolve.root=TRUE)
treeSegs0 = tree_layout(phy)
treeSegs1 = tree_layout(esophagus)
edgeMap = aes(x=xleft, xend=xright, y=y, yend=y)
vertMap = aes(x=x, xend=x, y=vmin, yend=vmax)
p0 = ggplot(treeSegs0$edgeDT, edgeMap) + geom_segment() + geom_segment(vertMap, data=treeSegs0$vertDT)
p1 = ggplot(treeSegs1$edgeDT, edgeMap) + geom_segment() + geom_segment(vertMap, data=treeSegs1$vertDT)
print(p0)
print(p1)
plot_tree(esophagus, "treeonly")
plot_tree(esophagus, "treeonly", ladderize="left")
```

UniFrac *Calculate weighted or unweighted (Fast) UniFrac distance for all sample pairs.*

Description

This function calculates the (Fast) UniFrac distance for all sample-pairs in a `phyloseq-class` object.

Usage

```
UniFrac(physeq, weighted=FALSE, normalized=TRUE, parallel=FALSE, fast=TRUE)

## S4 method for signature 'phyloseq'
UniFrac(physeq, weighted = FALSE,
  normalized = TRUE, parallel = FALSE, fast = TRUE)
```

Arguments

`physeq`	(Required). `phyloseq-class`, containing at minimum a phylogenetic tree (`phylo-class`) and contingency table (`otu_table-class`). See examples below for coercions that might be necessary.
`weighted`	(Optional). Logical. Should use weighted-UniFrac calculation? Weighted-UniFrac takes into account the relative abundance of species/taxa shared between samples, whereas unweighted-UniFrac only considers presence/absence. Default is `FALSE`, meaning the unweighted-UniFrac distance is calculated for all pairs of samples.

`normalized`	(Optional). Logical. Should the output be normalized such that values range from 0 to 1 independent of branch length values? Default is `TRUE`. Note that (unweighted) `UniFrac` is always normalized by total branch-length, and so this value is ignored when `weighted == FALSE`.
`parallel`	(Optional). Logical. Should execute calculation in parallel, using multiple CPU cores simultaneously? This can dramatically hasten the computation time for this function. However, it also requires that the user has registered a parallel "backend" prior to calling this function. Default is `FALSE`. If FALSE, UniFrac will register a serial backend so that `foreach::%dopar%` does not throw a warning.
`fast`	(Optional). Logical. DEPRECATED. Do you want to use the "Fast UniFrac" algorithm? Implemented natively in the `phyloseq-package`. `TRUE` is now the only supported option. There should be no difference in the output between the two algorithms. Moreover, the original UniFrac algorithm only outperforms this implementation of fast-UniFrac if the datasets are so small (approximated by the value of `ntaxa(physeq) * nsamples(physeq)`) that the difference in time is inconsequential (less than 1 second). In practice it does not appear that this parameter should have ever been set to `FALSE`, and therefore the original UniFrac implementation perhaps never should have been supported here. For legacy code support the option is now deprecated here (the implementation was an internal function, anyway) and the `fast` option will remain for one release cycle before being removed completely in order to avoid causing unsupported-argument errors.

Details

`UniFrac()` accesses the abundance (`otu_table-class`) and a phylogenetic tree (`phylo-class`) data within an experiment-level (`phyloseq-class`) object. If the tree and contingency table are separate objects, suggested solution is to combine them into an experiment-level class using the `phyloseq` function. For example, the following code

```
phyloseq(myotu_table, myTree)
```

returns a `phyloseq`-class object that has been pruned and comprises the minimum arguments necessary for `UniFrac()`.

Parallelization is possible for UniFrac calculated with the `phyloseq-package`, and is encouraged in the instances of large trees, many samples, or both. Parallelization has been implemented via the `foreach-package`. This means that parallel calls need to be preceded by 2 or more commands that register the parallel "backend". This is acheived via your choice of helper packages. One of the simplest seems to be the *doParallel* package.

For more information, see the following links on registering the "backend":

foreach package manual:

`http://cran.r-project.org/web/packages/foreach/index.html`

Notes on parallel computing in `R`. Skip to the section describing the *foreach Framework*. It gives off-the-shelf examples for registering a parallel backend using the *doMC*, *doSNOW*, or *doMPI* packages:

`http://trg.apbionet.org/euasiagrid/docs/parallelR.notes.pdf`

Furthermore, as of `R` version `2.14.0` and higher, a parallel package is included as part of the core installation, `parallel-package`, and this can be used as the parallel backend with the `foreach-package` using the adaptor package "doParallel". `http://cran.r-project.org/web/packages/doParallel/index.html`

See the vignette for some simple examples for using doParallel. `http://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf`

UniFrac-specific examples for doParallel are provided in the example code below.

Value

a sample-by-sample distance matrix, suitable for NMDS, etc.

References

`http://bmf.colorado.edu/unifrac/`

The main implementation (Fast UniFrac) is adapted from the algorithm's description in:

Hamady, Lozupone, and Knight, "Fast UniFrac: facilitating high-throughput phylogenetic analyses of microbial communities including analysis of pyrosequencing and PhyloChip data." The ISME Journal (2010) 4, 17–27.

See also additional descriptions of UniFrac in the following articles:

Lozupone, Hamady and Knight, "UniFrac - An Online Tool for Comparing Microbial Community Diversity in a Phylogenetic Context.", BMC Bioinformatics 2006, 7:371

Lozupone, Hamady, Kelley and Knight, "Quantitative and qualitative (beta) diversity measures lead to different insights into factors that structure microbial communities." Appl Environ Microbiol. 2007

Lozupone C, Knight R. "UniFrac: a new phylogenetic method for comparing microbial communities." Appl Environ Microbiol. 2005 71 (12):8228-35.

See Also

`distance`

`unifrac` in the picante package.

Examples

```
################################################################################
# Perform UniFrac on esophagus data
################################################################################
data("esophagus")
(y <- UniFrac(esophagus, TRUE))
UniFrac(esophagus, TRUE, FALSE)
UniFrac(esophagus, FALSE)
# ################################################################################
# # Now try a parallel implementation using doParallel, which leverages the
# # new 'parallel' core package in R 2.14.0+
# # Note that simply loading the 'doParallel' package is not enough, you must
# # call a function that registers the backend. In general, this is pretty easy
# # with the 'doParallel package' (or one of the alternative 'do*' packages)
# #
# # Also note that the esophagus example has only 3 samples, and a relatively small
# # tree. This is fast to calculate even sequentially and does not warrant
# # parallelized computation, but provides a good quick example for using UniFrac()
# # in a parallel fashion. The number of cores you should specify during the
# # backend registration, using registerDoParallel(), depends on your system and
# # needs. 3 is chosen here for convenience. If your system has only 2 cores, this
# # will probably fault or run slower than necessary.
# ################################################################################
```

```
# library(doParallel)
# data(esophagus)
# # For SNOW-like functionality (works on Windows):
# cl <- makeCluster(3)
# registerDoParallel(cl)
# UniFrac(esophagus, TRUE)
# # Force to sequential backed:
# registerDoSEQ()
# # For multicore-like functionality (will probably not work on windows),
# # register the backend like this:
# registerDoParallel(cores=3)
# UniFrac(esophagus, TRUE)
################################################################################
```

`[,otu_table,ANY,ANY,ANY-method` *Method extensions to extraction operator for phyloseq objects.*

Description

See the documentation for the `Extract` generic, defined in the R `base-package` for the expected behavior.

Usage

```
## S4 method for signature 'otu_table,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'sample_data,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'taxonomyTable,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'XStringSet,character,ANY,ANY'
x[i]
```

Arguments

`x`	object from which to extract element(s) or in which to replace element(s).
`i`	indices specifying elements to extract or replace. Indices are `numeric` or `character` vectors or empty (missing) or `NULL`. Numeric values are coerced to integer as by `as.integer` (and hence truncated towards zero). Character vectors will be matched to the `names` of the object (or for matrices/arrays, the `dimnames`): see ‘Character indices’ below for further details. For `[`-indexing only: `i`, `j`, `...` can be logical vectors, indicating elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. `i`, `j`, `...` can also be negative integers, indicating elements/slices to leave out of the selection. When indexing arrays by `[` a single argument `i` can be a matrix with as many columns as there are dimensions of `x`; the result is then a vector with elements corresponding to the sets of indices in each row of `i`. An index value of `NULL` is treated as if it were `integer(0)`.

`j`	See `Extract`
`...`	See `Extract`
`drop`	For matrices and arrays. If `TRUE` the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement. See `drop` for further details.

Details

One special exception to standard behavior of these methods in phyloseq is that the `drop` argument is set internally to `FALSE`. This helps avoid bugs during complicated subsetting with multiple components, where it is necessary to be able to use a two dimensional indexing even if one of those dimensions has only 1 rank. Put another way, these phyloseq-defined extractions never collapse their result into a vector. See the documentation of `Extract` for more information about the `drop` argument.

See Also

`Extract`

Examples

```
data(esophagus)
nrow(otu_table(esophagus))
nrow(otu_table(esophagus)[1:5, ])
```

Index